CA 2E

Building Applications

Release 8.7

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

- Action Diagram Call Prompt Using an External name

  Naming a Function as an Action (see page 436) - Added the description for new parameters.

- Allow SQL Record Level Access

  YSQLFMT (see page 54) - Added a model value.

- Device User Source Substitution Variables

  Substitution Variables (see page 400) - Added a set of new substitution variables for the device user source.

- Model Values Used in Building Functions (see page 41) - Updated model values.

- Refresh Action Diagram Statements

  YRFSACT (see page 50) - Added a new controlling model value named YRFSACT.

- Suppress Display of NLL parameters

  Specifying Parameters for an Action Function (see page 438) - Added the description about toggling F15 to display parameters other than the NLL parameters.

- Trigger Error Processing if no Control Data

  Trigger Router (see page 234) - Updated with two new messages information.

- Allow RLA Access over DDL Database
  - Acronyms (see page 26)
  - YDBFGEN (see page 45)
  - DSPTRN Device Function (see page 108)
  - EDTTRN Device Function (see page 125)
  - Generation Mode (see page 245)
  - Distributed File I/O Control (see page 249)
- Allow SQL/DDL generation without hard-coded schema name
  - YSQLCOL (see page 53)
- Allow LVLCHK(*YES) for SQL/DDL indexes having RCDFMT keyword
  - YSQLFMT (see page 54)
  - YLVLCHK (see page 48)

- YSQLFMT override
  - YSQLFMT (see page 54)
- Meaningful Names for SQL/DDL
  - YSQLVNM (see page 55)
- Option to Generate RLA against DDL
  - YDDLDBA (see page 45)
- Protect Trigger loss when moving to DDL/SQL
  - CA 2E Trigger Limitations (see page 223)
- Corrected Function Structure Charts
  - Change Object (see page 654)
  - Delete Object (see page 656)
  - Display File (Chart 3 of 5) (see page 659)
  - Display Record (Chart 1 of 5) (see page 662)
- IBM Limitation - File name is valid system name
  - YSQLVNM (see page 55)
- Meaningful Names for SQL/DDL--Control Table vs Fields
  - YSQLVNM (see page 55)
  - YDDLDBA (see page 45)
- Change the special value from *S to *I
  - Naming a Function as an Action (see page 436)

Contents

Chapter 3: Defining Functions 61

Chapter 7: Modifying Function Options 237

Chapter 1: An Introduction to Functions

This chapter provides an overview to *Building Applications*. Its purpose is to help you understand the CA 2E (formerly known as CA 2E) concepts for using functions in your model. This guide provides you with instructions on building functions in CA 2E including:

- Setting system default values
- Defining, copying, documenting, generating and compiling, and deleting functions
- Modifying function options, function parameters, device designs, and action diagrams
- Tailoring functions for performance

Each chapter is designed to provide you with the information you need to perform the identified task. Review the entire guide or see the chapter relating to the specific task you want to perform.

This section contains the following topics:

Organization

This chapter provides you with a high-level overview of the CA 2E concepts for building functions. The remaining chapters contain conceptual material and instructions on the specific tasks required to complete each step of the process.

Where necessary, these chapters also reference other topics and chapters in this guide or other guides containing related material.

We recommend that before you build your functions, you read or review the material in the following CA 2E guides:

- Overview
- Implementation
- Defining a Data Model
- Building Access Paths
- Generating and Implementing Applications

Terms Used in This Module

Descriptions of the acronyms, values, and abbreviations used in this guide are defined here and again the first time they are appearing in text. Thereafter, only the acronym, value, or abbreviation is used.

Acronyms

The following acronyms appear in this guide:

Acronym	Meaning
CBL	COBOL
CL	Control Language
CSG	Client Server Generator
DDL	Data Definition Library
DDS	Data Description Specifications
DRDA	Distributed Relational Database Architecture
ESF	External Source Format
HLL	High Level Language
IBM	International Business Machines Corporation
NPT	Non-Programmable Terminal
OS	Operating System
RPG	Report Program Generator
SAA	Systems Application Architecture
SQL	Structured Query Language
UIM	User Interface Manager

Values

The following values appear in this guide:

Value	Meaning
CPT	Capture File
PHY	Physical Access Path
QRY	Query Access Path
REF	Reference File
RSQ	Resequence Access Path
RTV	Retrieval Access Path
SPN	Span Access Path
STR	Structure File
UPD	Update Access Path

Abbreviations

The following abbreviations appear in this guide:

Abbreviation	Meaning
CHGOBJ	Change object
CNT	Count
CRTOBJ	Create Object
DLTOBJ	Delete Object
DRV	Derived
DSPFIL	Display File
DSPRCD	Display Record
DSPRCD2	Display Record 2
DSPRCD3	Display Record 3
DSPTRN	Display Transaction
EDTFIL	Edit File
EDTRCD	Edit Record
EDTRCD2	Edit Record 2

Abbreviation	Meaning
EDTRCD3	Edit Record 3
EDTTRN	Edit Transaction
EXCEXTFUN	Execute External Function
EXCINTFUN	Execute Internal Function
EXCMSG	Execute Message
EXCUSRPGM	Execute User Program
EXCUSRSRC	Execute User Source
MAX	Maximum
MIN	Minimum
PMTRCD	Prompt Record
PRTFIL	Print File
PRTOBJ	Print Object
RTVMSG	Retrieve Message
RTVOBJ	Retrieve Object
SELRCD	Select Record
SNDCMPMSG	Send Completion Message
SNDERRMSG	Send Error Message
SNDINFMSG	Send Information Message
SNDSTSMSG	Send Status Message
SUM	Sum
USR	User

Understanding Functions

A function defines a process that operates on files and fields in your database. CA 2E allows you to link functions together to create larger processes that become the building blocks of your application. You can link functions together as components to define an application system. You can implement several separate functions as a single HLL program. There are two ways to implement a function:

- External—the function is implemented as a separate HLL program
- Internal— the function is implemented as source code within the calling function

Function Types

There are a number of different function types that fall into the following four classes:

- Standard functions
- Built-In functions
- Function fields
- Message functions

Standard Functions

Standard functions specify entire programs or subroutines. User-defined processing can be specified to take place at appropriate points within all standard functions. Standard functions are intended to provide ready-made building blocks that, when put together, make up your application system. The standard functions are divided into the categories described below.

Database Functions

Database functions specify basic routines for updating the database. There are four different database functions, each defining a subroutine to either create, change, delete, or retrieve data. Database functions are implemented as part of an external standard function. All database functions are internal functions. Once you define a database function you can use it in many different functions.

The database functions are:

- Create Object (CRTOBJ)
- Change Object (CHGOBJ)
- Delete Object (DLTOBJ)
- Retrieve Object (RTVOBJ)

For more information on database functions, see the chapter Defining Functions (see page 61).

Device Functions

Device functions specify interactive programs of a number of types and report programs. These programs consist of either a panel design or report design and an action diagram. Device functions are external functions with the exception of Print Object (PRTOBJ), which is an internal function. You implement device functions as programs that operate over databases. The device functions are:

- Display Record (DSPRCD)
- Display Record 2 panels (DSPRCD2)
- Display Record 3 panels (DSPRCD3)
- Prompt Record (PMTRCD)
- Edit Record (EDTRCD)
- Edit Record 2 panels (EDTRCD2)
- Edit Record 3 panels (EDTRCD3)
- Display File (DSPFIL)
- Edit File (EDTFIL)
- Select Record (SELRCD)
- Display Transaction (DSPTRN)
- Edit Transaction (EDTTRN)
- Print File (PRTFIL)
- Print Object (PRTOBJ)

For more information on device functions, see the chapter Defining Functions (see page 61).

User Functions

User functions specify additional building blocks of user-written processing. User functions provide a means of incorporating user programs and subroutines into CA 2E generated applications. Their processing steps can be specified with action diagrams or user-written HLL. They can be implemented as inline code (internal functions) or calls to separate programs (external functions). The user functions are:

- Execute Internal Function (EXCINTFUN)
- Execute External Function (EXCEXTFUN)
- Execute User Program (EXCUSRPGM)
- Execute User Source (EXCUSRSRC)

For more information on user functions, see the chapter Defining Functions (see page 61).

Built-In Functions

Built-in functions execute common low-level functions such as arithmetic operations, character string manipulations, and control operations such as commitment control and program exit. Built-in functions are specified within action diagrams and are implemented as inline source code within calling functions.

The following table consists the list of built-in functions:

Function	Meaning
*ADD	Add
*COMMIT	Commit
*COMPUTE	Compute
*CONCAT	Concatenation
*CVTVAR	Convert Variable
*DATE DETAILS	Date Details
*DATE INCREMENT	Date Increment
*DIV	Divide
*DIV WITH REMAINDER	Divide with Remainder
*DURATION	Date Duration
*ELAPSED TIME	Elapsed Time
*EXIT PROGRAM	Exit Program
*MODULO	Modulo
*MOVE	Move
*MOVE ALL	Move All
*MOVE ARRAY	Move array subfield
*MULT	Multiply
*QUIT	Quit
*ROLLBACK	Rollback
*RTVCND	Retrieve Condition
*RTVFLDINF	Retrieve Field Information
*SET CURSOR	Set Cursor
*SUB	Subtract
*SUBSTRING	Substring

Function	Meaning
*TIME DETAILS	Time Details
*TIME INCREMENT	Time Increment

For more information on built-in functions see the chapter Modifying Action Diagrams (see page 431).

Function Fields

A function field is a field whose value is not physically stored in the database, but is derived from other fields or files. Function fields are always associated with only one result parameter, the derived field itself, along with a variable number of input parameters that are used to derive the calculation.

CA 2E also provides a number of ready-made function fields such as summing or counting that you can call from within a function. Once the function field is defined, CA 2E automatically incorporates its associated processing logic when it is used. The function fields are:

- Sum (SUM)
- Maximum (MAX)
- Minimum (MIN)
- Count (CNT)
- Derived (DRV)
- User (USR)

For more information on function fields refer to the chapter Defining Functions (see page 61).

Message Functions

Message functions define messages that you want to appear at a workstation using special CA 2E facilities, or they define other variables for use by the function. Message functions are specified in a similar way to other function types, but are implemented using i OS message descriptions and are sent by a call to a standard CL subroutine. Their implementation as i OS message descriptions allows them to be changed for translation to another national language without affecting the calling program. They can make direct references to fields in your data model. You can also use Message functions to execute i OS command requests. The message functions are:

- Send Error Message (SNDERRMSG)
- Send Information Message (SNDINFMSG)

- Send Complete Message (SNDCMPMSG)
- Send Status Message (SNDSTSMSG)
- Retrieve Message (RTVMSG)
- Execute Message (EXCMSG)

For more information on message functions refer to the chapter Defining Functions (see page 61).

Basic Properties of Functions

CA 2E functions have the following properties.

Function Names

The name of each function can contain up to 25 characters including any embedded blanks, and must be unique within a given file.

Function Components

Functions generally consist of the following components: function options, parameters, device designs, and action diagrams.

Function Options

Function options enable you to customize the features of your functions including database changes, display features, exit control, commitment control, exception routines, generation options, and environment options.

For more information on function options, refer to the chapter Modifying Function Options (see page 237).

Parameters

Parameters specify which field values will be passed into the function at execution and which fields will be returned from the function on completion. In addition, parameters are used to define local variables for the function.

For more information on parameters, see the chapter Modifying Function Parameters (see page 257).

Device Designs

Device designs specify the visual presentation of the two types of devices used by functions:

- Panel (display)
- Report

For more information on device designs, see the chapter Modifying Device Designs (see page 287).

Action Diagrams

Action diagrams specify the processing steps for the program function logic. This is a combination of default (CA 2E supplied) logic and optional user-defined processing logic.

The following table shows which component applies to each function type:

Function Class	Parameters	Device Design	Action Diagrams	Function Options
Device Functions	Y	Y	Y	Y
Database Functions	Y	N	N	Y,N
User Functions	Y	N	Y,N **(1)**	Y,N **(3)**
Messages	Y	N	N	N
Function Fields	Y	N	Y, N **(2)**	N
Built-in Functions	Y	N	N	N

1. \EXCINTFUN and EXCEXTFUN have action diagrams. EXCUSRSRC and EXCUSRPGM do not have action diagrams.
2. Only DRV function fields have action diagrams; all other function fields do not.
3. EXCUSRSRC is the only user function that has no function option.

For more information on action diagrams, see the chapter Modifying Action Diagrams (see page 431).

Default Device Function Processing

A default device function generates and compiles into a working program with a default device design and a default action diagram. Additional logic is required only to achieve the specific functionality required for the application. User points are provided in the default action diagram where the logic can be inserted. You can also make changes to the default device design, parameters, and function options.

In addition to the working program, default device design, and action diagram, CA 2E provides default processing such as file-to-file validation, database checking, and prompt logic.

For more information:

- On the action diagram editor, see the chapter Modifying Action Diagrams (see page 431).
- On editing the device design, see the chapter Modifying Device Designs (see page 287).
- On standard functions, see the chapters Defining Functions (see page 61) and Modifying Function Options (see page 237).

Functions and Access Paths

Functions that operate on a file are always attached to the file by an access path. Records are automatically read from the file using the access path, which specifies the order and selection criteria in which records from the file are processed. Since the access path can be based on several files, a function can process data from more than one file. In addition, a generated program can be composed of several functions, each processing different access paths. Default panel and report formats are derived from the function's access path.

For more information:

- Function types, see the chapter Defining Functions (see page 61).
- Access paths, see the *Building Access Paths Guide*.

Additional Processing

CA 2E automatically supplies additional processing logic when building functions in your model including integrity checking, validating data entered in the fields, and linking functions.

Integrity Checking

CA 2E automatically includes logic to perform domain and referential integrity checking in the default processing of the interactive function types.

Domain Integrity Checking

Domain integrity checking ensures that when a field is used in place of another field, these fields are similar. This is enforced by ensuring that the fields share the same reference fields. Fields that have the same domain have the same set of allowed values for conditions.

When assigning parameters to a function within the action diagram editor, CA 2E verifies that the field you are passing and the field that is specified as the input parameter have the same domain. Fields of the same type and length do not necessarily have the same domain. Domains can be shared by fields through referencing (REF). If the domains do not match, you receive a warning message. To ignore the warning, press Enter.

Referential Integrity Checking

This check ensures that the relations specified in the model are satisfied. For example, if you specify the relation Order Refers to Customer, the HLL source code generated to implement a maintenance function on the Order file includes a read to the Customer file to check that a record for the specified Customer Code exists. You can adjust the actual referential integrity checking that is performed in any given function.

For more information on relations, refer to the the chapter "Understanding Your Data Model" in the *Defining a Data Model* guide.

Field Validation

Validation attributes specify how data entered into the field is to be validated. Validation includes both the attributes supported by i OS, such as uppercase lowercase checking, modulus checking, i OS valid name checking, and validation through a check condition. You can define additional field validation logic for any field type and automatically incorporate it in any function using the field.

For more information about field type, see Defining a Data Model, Using Fields topic in the chapter "Understanding Your Data Model."

Linking Functions

CA 2E automatically links certain functions together. For external update functions, CA 2E automatically links to the database functions that update data. CA 2E also automatically provides the linkage to functions that allow lookup capabilities. You can use action diagrams to specify further links between functions.

For more information on action diagrams, refer to the chapter Modifying Actions Diagrams (see page 431).

When HLL code is generated to implement several connected internal functions, such as functions that are implemented as inline code, the HLL used to generate the functions is determined by the source type of the external function, which calls the internal function or functions.

If you connect Execute User Source (EXCUSRSRC) functions together with other functions, you must ensure that the connected functions all have the same HLL implementation types (that they are all RPG or all COBOL functions.)

Although CA 2E does not impose any limitation on the recursive linking of external functions, some high-level languages do. For example, the same RPG program may not appear twice in a job's invocation stack. This means that you should avoid having a function calling itself, either directly or via another function.

Building Block Approach, an Overview

Building an application is a matter of defining or choosing the right functions and putting them together to meet your requirements. CA 2E functions serve as the components for applications. When implemented, several functions may work together as a single HLL program. Functions can also call other functions, based on default connections or action diagram specifications.

The process of designing your application should include the step of breaking down your operations into simple building blocks. Each CA 2E function performs a unique, defined task. Correlating your operations with these functions is called function normalization. The structure of CA 2E provides the means for normalizing functions and for constructing more complex functions from the simple building blocks.

Function normalization encourages the use of a single function to perform a single defined action. More complex functions can then be constructed by linking together lower level functions. This approach allows for development and testing to be incremental and reduces the overall development and maintenance effort.

For example, to construct a routine to calculate the days between two dates you should first construct a function to convert a date into one absolute day number. You can then use this function to convert the from and to dates to an equivalent numeric value. Then you can use subtraction to yield the difference. This same low level function can also be used in other functions that add, drop, or subtract days from a date, without the need to redevelop or repeat logic.

The CA 2E building block approach gives you categories of functions, and each category has a specific implementation as follows:

- **Standard device functions**—Specify interactive or report programs. These functions have device designs attached to them. These functions work together with the database functions to view, add, change, or delete data in your files.
- **Internal functions**—Are implemented as source code within calling functions.
- **External functions**—Are implemented as HLL programs such as batch processing and device functions.
- **Built-in functions**—Execute common low-level functions and such tasks as arithmetic operations and commitment control.
- **Function fields**—Specify how to calculate derived fields. A derived field is any field with a value that is calculated from other fields when accessed in a routine, rather than physically stored in the database.
- **User-written functions**—Specify user-written processes with either action diagrams or RPG or COBOL subroutines or programs.

For more information:

- About adding functions, see the chapter Defining Functions (see page 61).
- About action diagrams, see the chapter Modifying Action Diagrams (see page 431).

Top-Down Application Building

If you are developing a new application, a top-down approach is a good way to design the functions for your application. This approach, which assumes that your data model and access paths are defined, includes

- Identifying the functions to be called from points in processing
- Working top-down to define functions and function parameters as needed

- Specifying top level constructs and the logic flow of user points
- Filling in user point details

For more information about:

- The functions you will select for your application, see the chapter Defining Functions (see page 61).
- Getting the best performance from your application, see the chapter Tailoring for Performance (see page 631).

Chapter 2: Setting Default Options for Your Functions

This chapter identifies the model values specific to functions and shows you how to change them, how to change the default names that CA 2E assigns to functions, and function key defaults.

This section contains the following topics:

Model Values Used in Building Functions

This topic covers the model values used by functions. Function options can affect the device design and processing defaults. Model values are shipped as defaults for the Create Model Library (YCRTMDLLIB) command.

Many function options are derived from model values. If you find that you often change these options at the function level, you may want to review the settings in your model and change them at the model level.

For more information about:

- Model values you can change at the function level, see the section Changing Model Values (see page 57).
- Descriptions of each model value, YCHGMDLVAL, see the *Command Reference* guide.

YABRNPT

The YABRNPT value is only for NPT generation, and enables you to choose between creations of CA 2E Action Bars or DDS Menu Bars for a given function. The default is DDS Menu Bars for models created as of r5.0 of COOL:2E. For existing models upgraded to r5.0, the default is Action Bars.

We recommend that you migrate to DDS Menu Bars over time since DDS Menu Bars make use of the new i OS ENPTUI features, which allow the menu bars to be coded in the DDS for the display file. The CA 2E Action Bars require that an external program be called to process the action bar. As a result, the DDS Menu Bars are faster, have more functionality, and create more efficient CA 2E functions.

For more information about NPT user interface options, see ENPTUI in the chapter Modifying Device Designs (see page 287).

YACTCND

The Action Diagram Compound Symbols (YACTCND) model value defines the symbols used in editing and displaying compound condition expressions.

The format for modifying this design option is:

```
YCHGMDLVAL  MDLVAL(YACTCND)  VALUE('& AND | OR ^ NOT  ( ( ) ) c c')
```

For more information about compound conditions, see Entering and Editing Compound Conditions in the chapter Modifying Action Diagrams (see page 431).

YACTFUN

The Action Diagram Compute Symbols (YACTFUN) model value defines the symbols used in editing compute expressions, which include + - * / \ ( ) x. You are only likely to change these defaults if you have national language requirements. The binary code values for these symbols can map to different values, depending on the code page in use. For example, a forward slash (/) on the US code page would map to a cedilla in a French National code page.

For more information on compute expressions, see Entering and Editing Compound Conditions in the the chapter Modifying Action Diagrams (see page 431).

YACTSYM

The Action Diagram Structure Symbols (YACTSYM) model value defines the symbols used in action diagrams. The shipped default is *SAA. The Action Diagram Editor and the Document Model Functions (YDOCMDLFUN) command use this design option.

YACTUPD

The Action Diagram Update (YACTUPD) model value defines the default value for the Change/create function option on the Exit Function Definition panel. The shipped default is *YES. The value *CALC sets the Change/create function option to Y only when a change to the function's action diagram or panel design is detected.

YALCVNM

The Automatic Name Allocation (YALCVNM) model value indicates whether CA 2E should automatically allocate DDS and object names. The shipped default is *YES.

For more information on name allocation, see the *Implementation Guide.*

YBNDDIR

Specifies a binding directory that can resolve the location of any previously compiled RPGIV modules. Use this model value while compiling RPGIV programs with the CRTBNDRPG command.

Note: For more information, see the section *The YBNDDIR Model Value* in the Chapter *ILE Programming*.

YCNFVAL

The Confirm Value (YCNFVAL) model value determines the initial value for the confirm prompt. The shipped default is *NO.

For more information on function options, see the chapter Modifying Function Options (see page 237).

YCPYMSG

The Copy Back Messages (YCPYMSG) model value specifies whether, at program termination, outstanding messages on the program message queue are copied to the message queue of the calling program. The shipped default is *NO.

For more information on function options, see the chapter Modifying Function Options (see page 237).

YCRTENV

The Creation Environment (YCRTENV) model value determines the environment in which you intend to compile source is the iSeries. The shipped default is the iSeries.

For more information about:

- Controlling design, see the *Implementation Guide.*
- Environments, see the *Generating and Implementing Applications* guide, in the chapter "Preparing for Generation and Compilation".

YCUAEXT

The CUA Device Extension (YCUAEXT) model value determines whether the text on the right side text is used for device designs. The shipped default is *DEFAULT, which results in no right text and no padding or dot leaders.

The YCUAEXT value, *C89EXT (for CUA Text), provides CUA design features on top of those which the model value YSAAFMT provides, such as defaulting and alignment of right side text, padding or dot leaders to connect fields with field text, and prompt instruction lines on all device function types.

For more information on field attributes and right side text defaults, see the chapter, "Modifying Device Designs," Device Design Conventions and Styles.

YCUAPMT

The CUA Prompt (YCUAPMT) model value controls the CUA prompt (F4). If enabled, this design option enables end users to request a list display of allowed values by pressing F4. The value *CALC provides additional F4 functionality by processing the CALC: user points in the function where F4 is pressed—for example, to provide Retrieve Condition functionality.

The default value for YCUAPMT is *MDL. This value directs CA 2E to enable the CUA prompt at the model level if the YSAAFMT model value is *CUATEXT or *CUAENTRY.

For more information about:

- Setting display defaults, see the chapter, "Modifying Device Designs"
- On the *CALC value, see the *Command Reference*, the YCHGMDLVAL command

YCUTOFF

The Year Cutoff (YCUTOFF) model value specifies the first of the hundred consecutive years that can be entered using two digits. It is specified as 19YY, which represents the hundred years: 19YY to 20YY-1. Values between YY and 99 are assumed to be in the 20th century; namely, 19YY to 1999; values between 00 and YY-1 are assumed to be in the 21st century; namely 2000 to 20YY-1. The default is 1940. The YCUTOFF value is retrieved at run time and applies to all date field types: DTE, D8#, and DT#.

YDATFMT

The Date Format (YDATFMT) model value works in conjunction with the model value YDATGEN. If YDATGEN is *VRY. The setting for YDATFMT determines the order of the date components at run time; for example, MMDDYY or DDMMYY.

YDATGEN

The Date Validation Generation (YDATGEN) model value determines the type of date editing source code CA 2E generates. With YDATGEN set to *VRY, you can change the date format for an application with the YDATFMT model value. No recompilation of functions is necessary.

YDBFGEN

The Database Implementation (YDBFGEN) model value defines the method for database file generation and implementation: DDS, SQL or DDL.

YDDLDBA

The Database Access Method (YDDLDBA) model value specifies a method of accessing the database (RLA or SQL) when a function's Generation Mode option is set to A(ACPVAL) or M(MDLVAL), which resolves to DDL type.

***RLA**

Specifies that the external function generates with RLA access.

***SQL**

Specifies that the external function generates with SQL access.

Note: To generate or regenerate a function with RLA code for DDL database, set the YSQLVNM model value to *DDS or *LNG or *LNT, or *LNF and set the YDDLDBA model value to *RLA.

YDFTCTX

The Parameter Default Context (YDFTCTX) model value specifies the default context to use for a given function call in the action diagram editor when no context is supplied: LCL or WRK. The shipped default is *WRK.

YDSTFIO

The Distributed File I/O Control (YDSTFIO) model value, together with model value YGENRDB, provides DRDA support. The shipped default value is *NONE, indicating that CA 2E will not generate distributed functionality.

For more information on DRDA, see Generating and Implementing Applications in the chapter "Distributed Relational Database Architecture."

YERRRTN

For RPG-generated functions, the Error Routine (YERRRTN) indicates whether CA 2E will generate an error handling routine (*PSSR) in the program that implements the function. The shipped default value is *NO.

Note: For EXCUSRPGM functions, this value specifies whether an error-handling routine should be generated in the calling program to check the value of the *Return code on return from the EXCUSRPGM (if the EXCUSRPGM does not have the *Return code as a parameter, this check will not be generated).

YEXCENV

The call to a CL program that implements an EXCMSG function uses an i OS program. The Execution Environment (YEXCENV) model value determines the default environment, QCMD (i OS), in which Execute Message (EXCMSG) functions execute.

For more information about:

- EXCMSG functions, see Function Types, Message Types, and Function Fields in the chapter, "Defining Functions"
- QCMD and QCL, see Generating and Implementing Application*s*— Managing Your Work Environment in the chapter "Preparing for Generation and Compilation"

YGENCMT

The time required to generate a function can be significantly improved if comments are not required for the generated source code. The YGENCMT model value lets you specify whether or not comments are placed in the resulting generated source code. You can specify that all comments (*ALL), 'standard' comments (*STD), only header comments (*HDR), or no comments (*NO) be generated. The shipped default is *ALL.

YGENHLP

The Generate Help Text (YGENHLP) model value allows you to specify whether help text is generated for a particular function. You can specify generation of the function only (*NO), help text only (*ONLY), or both the function and help text (*YES). This value can be overridden at the function level. The shipped default is *YES.

YGENRDB

The Generation RDB Name (YGENRDB) model value provides the DRDA support for specifying a default database. When you execute the CRTSQLxxx command, this database is used in creation of the SQL package. The default value for YGENRDB is *NONE, which means that DRDA compilation is not enabled.

For more information about DRDA, see Generating and Implementing Applications in the chapter "Distributed Relational Database Architecture."

YHLLGEN

The HLL to Generate (YHLLGEN) model value identifies the default HLL type for new functions. The HLLGEN parameter on YCRTMDLLIB sets this model value.

Note: To default to the value for model value YSYSHLL, select *SYSHLL for the parameter HLLGEN.

YHLLVNM

The HLL Naming Convention (YHLLVNM) model value determines the HLL conventions for new function names. The HLLVNM parameter on YCRTMDLLIB sets this model value. The default is *RPGCBL, allocation of names that both RPG and COBOL compilers support.

For more information about converting HLLs, see Generating and Implementing Applications—Converting a Model from One HLL to Another, in the chapter "Preparing for Generation and Compilation."

YHLPCSR

The Generate Cursor Sensitive Text (YHLPCSR) model value gives you the option of generating your function with cursor-sensitive help. That is help- specific to the context (cursor position) from which the end user requests it. The shipped default is Y (Yes).

YLHSFLL

The Leaders for Device Design (YLHSFLL) model value refers to the symbols to usedas leaders between text and input or output fields on panels. The shipped default value is *SAA, for SAA default left-hand filler characters. You can change any of these characters using the YCHGMDLVAL command.

YLVLCHK

The Generate IDX with LVLCHK(*YES) (YLVLCHK) model value specifies whether an Index (SQL or DDL), when generated with the RCDFMT keyword in it, is created with LVLCHK(*YES). The possible values are *NO and *YES. The shipped default is *NO.

If YLVLCHK is specified as *NO, then existing defaults around LVLCHK are retained when an SQL or DDL index is generated and created. The existing defaults for the LVLCHK attribute in the case of SQL and DDL are as follows.

- When a table or view is generated and created, it is generated with LVLCHK(*YES), irrespective of the existence of the RCDFMT keyword.
- When an index (SQL or DDL) is generated and created without the RCDFMT keyword, it is created with LVLCHK(*YES).
- When an index (SQL or DDL) is generated and created with the RCDFMT keyword, it is created with LVLCHK(*NO).

If YLVLCHK is specified as *YES (in addition to YSQLFMT set as *YES), upon subsequent generation and creation of an index (SQL or DDL), the index is created with LVLCHK(*YES).

Note: If YLVLCHK is set to *YES (along with YSQLFMT set to *YES), upon re-generation of the access path, an additional line "Y* CHGLF LVLCHK(*YES)" is generated into the header portion. This informs YEXCSQL to create the corresponding index with LVLCHK(*YES). For any other combination of YLVLCHK and YSQLFMT, there is no change to the existing processing.

YNPTHLP

The NPT Help Default Generation Type (YNPTHLP) model value determines the type of help text to generate for NPT functions. All CA 2E functions are NPT unless the functions are being generated for a GUI product. The types are UIM or TM. The shipped default for YNPTHLP is *UIM.

For more information about UIM support, see Objects from UIM Generation in the chapter "Implementing Your Application."

YNLLUPD

The Null Update Suppression (YNLLUPD) model value sets the default for whether CHGOBJ functions update or release the database record if the record was not changed. This can be overridden with a matching function option. The shipped default is *NO.

- *NO

  CHGOBJ functions do not check whether the record has changed before updating the database. In other words, null update suppression logic is not generated in CHGOBJ functions.

- *AFTREAD

  CHGOBJ checks whether the record changed between the After Data Read and Data Update user points.

- *YES

  CHGOBJ checks whether the record changed both after the Data Read and after the Data Update' user points.

For more information about:

- CHGOBJ database function, refer to the chapter, "Defining Functions"
- Suppressing null updates, see Understanding Contexts, PGM in the chapter "Modifying Action Diagrams"

YOBJPFX

The Member Name Prefix (YOBJPFX) model value specifies the prefix (up to two characters) CA 2E uses to generate object names. The shipped default is UU. If you change this prefix, do not use Q, #, and Y because they are reserved characters for CA 2E.

For more information about naming prefixes, see the *Implementation Guide*.

YPMTGEN

The Prompt Implementation (YPMTGEN) model value specifies whether the text on your device designs is generated, implemented, and stored in a message file, making it available for national language translation. The shipped default value is *OFF. The parameter PMTGEN on the YCRTMDLLIB command initially sets the YPMTGEN model value.

For more information about:

- National Language Support, see Generating and Implementing Applications in the chapter "National Language Support"
- YCRTMDLLIB, see the *Command Reference*

YPMTMSF

The Prompt Message File (YPMTMSF) model value specifies the message file into which device text message IDs are stored. CA 2E retrieves the messages from this message file at execution time.

For more information about National Language Support, see Generating and Implementing Applications in the chapter "National Language Support."

YPUTOVR

The DDS Put With Override (YPUTOVR) model value is a function generation option. It enables you to specify use of the DDS PUTOVR keyword in the generated DDS. This keyword, in effect, reduces the amount of data that needs transmission between the system and its workstations. Its use can improve performance, particularly on remote lines.

For more information about system performance, see the *IBM i Programming: Data Description Specifications Reference*.

YRFSACT

The Refresh Action Diagram on Entry (YRFSACT) model value specifies whether the YCHKFUNACT processor must be called during Action Diagram load to refresh the Action Diagram. The possible values are *NO and *YES. The shipped default is *NO.

Note: For more information about what is changed when an Action Diagram is refreshed, see the details of the YCHKFUNACT command in the *Command Reference Guide*.

YRP4HSP

Used by the RPGIV Generator for the contents of the Control (H) specification for objects of type *PGM. The allowed values are any RPGIV H-specification keywords, for example:

- DATEDIT(*YMD) DEBUG(*YES)
- DATFMT(*YMD)

Note: If you need to enter a value that is longer than 80 characters, you should use the command YEDTDTAARA DTAARA(YRP4HSPRFA)

YRP4HS2

Used by the RPGIV Generator for the contents of the Control (H) specification for objects of type *MODULE. The allowed values are any RPGIV H-specification keywords, for example:

- H DATFMT(*YMD)
- DATEDIT(*YMD) DEBUG(*YES)

Note: If you need to enter a value that is longer than 80 characters, you must use the command YEDTDTAARA DTAARA(YRP4HS2RFA)

YRP4SGN

The RPGIV generator includes some source generation options that you can set at a model level. These options are in the model value YRP4SGN in a data area called YRP4SGNRFA (RPGIV source generation options). YRP4SGNRFA is a 16-character data area.

Note: For more information, see the section *Model Value YRP4SGN* in the Chapter *ILE Programming*.

YSAAFMT

The SAA Format (YSAAFMT) model value controls the design standard for panel layout. This standard can be CUA. *CUAENTRY is the shipped default.

The DSNSTD parameter on the YCRTMDLLIB command controls the initial YSAAFMT value. You can override the header or footer for a function from the Edit Function Options panel. You can also change the value of YSAAFMT using the YCHGMDLVAL command.

For more information about:

- Using YSAAFMT options, see Device Design Conventions and Styles in the chapter "Modifying Device Designs"
- YSAAFMT values, see YCHGMDLVAL in the *Command Reference*

YSFLEND

The Subfile End (YSFLEND) model value controls whether the + sign or More. . . is displayed in the lower right location of the subfile to indicate that the subfile contains more records. This feature is available for all subfile functions. The shipped default is *PLUS. To change to *TEXT everywhere, change the model value and regenerate your subfile functions.

The setting of YSFLEND is resolved in the following areas:

- Generated applications
- Device designs
- Animated functions
- Function documentation (YDOCMDLFUN)

YSHRSBR

The Share Subroutine (YSHRSBR) model value specifies whether generated source code for subroutines are shared and whether the subroutine's interface is internal or external. This model value and its associated function option are available on the CHGOBJ, CRTOBJ, DLTOBJ, RTVOBJ, and EXCINTFUN function types.

YSNDMSG

For new functions, the Send Error Message (YSNDMSG) model value specifies whether to send an error message for only the first error found or for every error. In either case, outstanding messages clear when the end user presses Enter. The shipped default value is *NO, do not send all error messages; send only the first error message.

YSQLCOL

The Generate SQL Collection/Library Name (YSQLCOL) model value specifies whether a hard coded SQL Collection/Library name should be generated for tables, indexes and views. The possible values are *YES and *NO. The shipped default is *YES.

If YSQLCOL is specified as *YES, the SQL Collection/Library specified for the YSQLLIB model value is generated into the tables, indexes and views, by default, as is the case now. Subsequently when YEXCSQL is executed to create tables, indexes and views, they are created into the hard coded SQL Collection/Library. If YSQLCOL is specified as *NO, the SQL Collection/Library specified for the YSQLLIB model value is not generated into the tables, indexes and views. However, when YEXCSQL is executed subsequently, the tables, indexes and views are generated into the SQL Collection/Library specified for the YSQLLIB model value.

Note: If YSQLCOL is set to *NO and the access paths are generated, another change that can seen in the source, apart from the absence of hard coded SQL collection/Library name is, the previously generated Z* line "Z* YEXCSQL NAMING(*SQL)" is now generated as "Z* YEXCSQL NAMING(*SYS)".

YSQLFMT

The Generate SQL RCDFMT clause (YSQLFMT) model value specifies whether the RCDFMT keyword must be generated for SQL tables, views, and indexes. The possible values are *NO, and *YES. The shipped default is *NO.

If YSQLFMT is specified as *NO, the record format is the same as the table, index, or view name (if YSQLVNM (*DDS) is specified) or will be generated by the system (if YSQLVNM(*SQL) is specified). If YSQLFMT is specified as *YES, the RCDFMT value is calculated using the same rules as are used when DDS files are generated.

Note: Irrespective of the value of the YSQLFMT model value and if the generation mode is *DDL, the RCDFMT keyword is generated.

Important!

If YSQLFMT is set to *YES or *NO and a DDL index is generated and created, the index is created with LVLCHK(*NO).

If YSQLFMT is set to *YES and an SQL index is generated and created, the index is created with LVLCHK(*NO).

If YSQLFMT is set to *NO and an SQL index is generated and created, the index is created with LVLCHK(*YES).

If you want to change the LVLCHK attribute of the index to LVLCHK(*YES), the model value YLVLCHK must be set to *YES, and the corresponding index-related access path must be regenerated and re-created. Upon regeneration of the access path, an additional line "Y* CHGLF LVLCHK(*YES)" is generated in the header portion, which informs YEXCSQL to create the corresponding index with LVLCHK(*YES).

Note: If YSQLFMT is set to *YES, YLVLCHK is set to *YES and RUNSQLSTM is used to create an index (SQL or DDL), the index would still be created with LVLCHK(*NO). The current functionality does not cater to the RUNSQLSTM command. Tables and Views (SQL) and Tables (DDL) are created with LVLCHK(*YES) by default, irrespective of the YSQLFMT model value. Therefore, YSQLFMT and YLVLCHK model values have no effect on tables and views regarding the LVLCHK attribute.

YSQLLCK

The SQL Locking (YSQLLCK) model value specifies whether a row to be updated is locked at the time it is read or at the time it is updated. The default is *UPD, lock rows at time of update.

YSQLVNM

The SQL Naming (YSQLVNM) model value specifies whether to use the extended SQL naming capability. The valid values are:

***DDS**

Use DDS names. The shipped default.

***SQL**

Use the names of the CA 2E objects in the model.

***LNG**

Use the long names of the CA 2E objects in the model along with the DDS or implementation names.

***LNF**

Use the long field names of the CA 2E objects in the model along with the DDS or implementation names.

***LNT**

Use the long table names of the CA 2E objects in the model along with the DDS or implementation names.

Note:

If a table that has a valid system name (less than or equal to 10 bytes in length), is generated with YSQLVNM model value set as *LNG or *LNT, and when you set the Enhance SQL Naming option on the Edit File Details panel to **Y** and then generate the source, the table is created with (underscores) "_"s and "TABLE" as suffix, so that the name of the table becomes more than 10 char long.

Examples:

- CUSTOMER is generated as CUSTOMER_TABLE along with its 2E implementation name.
- CUST is generated as CUST__TABLE along with its 2E implementation name.
- C is generated as C_____TABLE along with its 2E implementation name.
- To generate or regenerate a function with RLA code for DDL database, set the YSQLVNM model value to *DDS or *LNG or *LNT, or *LNF and set the YDDLDBA model value to *RLA.

YSQLWHR

The SQL Where Clause (YSQLWHR) model value specifies whether to use OR or NOT logic when generating SQL WHERE clauses. The default is *OR.

For more information about the YSQLLCK and YSQLWHR model values, see the *Implementation Guide*.

YWSNGEN

The Workstation Generation (YWSNGEN) model value defines whether interactive CA 2E functions operate on non-programmable terminals (NPT) or on programmable workstations (PWS) communicating with an iSeries host. For programmable workstations, you also specify the PC runtime environment. YWSNGEN can be overridden by a function option. The possible values are:

- *NPT

  Generates CA 2E functions for non-programmable terminals (NPT) communicating with an iSeries host system.

- *GUI

  Generates CA 2E functions for non-programmable terminals together with a Windows executable running in a Windows environment under emulation to the host.

- *JVA

  Generates CA 2E functions for non-programmable terminals together with a Windows executable running in a Windows environment under emulation to the host and a Java executable running in a Windows environment using a Web browser with emulation to the host.

- *VB

  Generates CA 2E functions for non-programmable terminals together with a Visual Basic executable running in a Windows environment under emulation to the host.

User Interface Manager (UIM)

Three model values provide options for UIM help text generation:

- The Bidirectional UIM Help Text (YUIMBID) model value provides national language support of languages with both left-to-right and right-to-left orientations
- The Default UIM Format (YUIMFMT) model value provides paragraph or line tags
- The UIM Search Index (YUIMIDX) model value provides search for the index name derived from Values List prefix

Window Borders

Three model values provide design options for the appearance of the border on windows:

- The Window Border Attribute (YWBDATR) model value provides shadow or no shadow
- The Window Border Characters (YWBDCHR) model value provides dot/colon formation
- The Window Border Color (YWBDCLR) model value provides CUA default (Blue) or another color

For more information on Modifying Windows, see Editing Device Designs in the chapter Modifying Device Designs (see page 287).

Changing Model Values

This topic summarizes changing model values for a function of your model.

Function Level

You can override model value settings that determine function options at the function level from the Edit Function Options panel. You can reach this panel by zooming into the function from the Edit Functions panel, then pressing F7 (Options) from the Edit Function Details panel.

The model values that have corresponding fields on the Edit Function Options panel are:

Values	Meaning
YABRNPT	Create CA 2E Action Bars or DDS Menu Bars for NPT generation
YCNFVAL	Initial value for the confirm prompt
YCPYMSG	Copy back messages
YDBFGEN	Generation mode
YDSTFIO	Distributed file I/O control
YERRRTN	Generate error routine
YGENHLP	Generate help text
YNPTHLP	Type of help text to be generated
YPMTGEN	Screen text implementation

Values	Meaning
YSNDMSG	Send all error msgs (messages)
YSFLEND	Subfile end
YWSNGEN	Type of workstation

For more information about:

- Options applicable to each function see Function Types, Message Types, and Function Fields in the chapter Defining Functions (see page 61).
- On step-by-step procedures, see Specifying Function Option in the chapter Modifying Function Options (see page 237).

Model Level

You can change the setting of a model value for your model by executing the Change Model Value (YCHGMDLVAL) command. Be sure to use YCHGMDLVAL, rather than the i OS command, Change Data Area (CHGDTAARA). Changing model values involves more than changing data areas; many internal model changes are made by YCHGMDLVAL.

You should always exit from your model entirely when changing model values. Although the command can appear to run successfully while you are in the model, there is no guarantee that a full update has taken place.

For more information on using the YCHGMDLVAL command, see the *Command Reference* guide.

Changing a Function Name

To change a function name

1. Select the file. From the Edit Database Relations panel, type **F** next to the specific file and press Enter.

   The Edit Functions panel appears, listing the functions for that file.

2. Zoom into the function details. Type **Z** next to the specific function and press Enter.

   The Edit Function Details panel appears, showing the function name at the top.

3. Request to change the function name. Press F8 (Change name).

   The function whose name you want to change appears underlined on the panel.

4. Change the function name. Type the specific name. If you want, you can change any other underlined names to better correspond to the new function name . Press Enter, then F3 to exit.

Function Key Defaults

CA 2E assigns the standard function key usage of your design standard. You can specify additional function keys in action diagrams or modify existing function key default values.

For more information about function keys, see the chapter Modifying Device Designs (see page 287).

The following table shows the shipped device design defaults for the iSeries.

Meaning	iSeries default
*Help	F01/HELP
Prompt	F04
Reset	F05
*Change mode request	F09
*Change mode to Add	F09
*Change mode to Change	F09
*Delete request	F11
*Cancel	F12
*Exit	F03
*Exit request	F03
*Key panel request/*Cancel	F12
*IGC support	F18
Change RDB	F22
*Previous page request	F07/ROLLDOWN
*Next page request	F08/ROLLUP

The default is determined by the design standard selected. The iSeries default is used if the YSAAFMT model value is set to *CUATEXT or *CUAENTY.

Chapter 3: Defining Functions

This chapter is to describe the basic implementation of functions in CA 2E. The following information describes the various function types and gives a functional overview of what is involved in the function development process.

Before you define your functions, you should be familiar with the information in the following CA 2E guides:

- *Implementation*
- *Defining a Data Model*
- *Building Access Paths*

This section contains the following topics:

Navigational Techniques and Aids

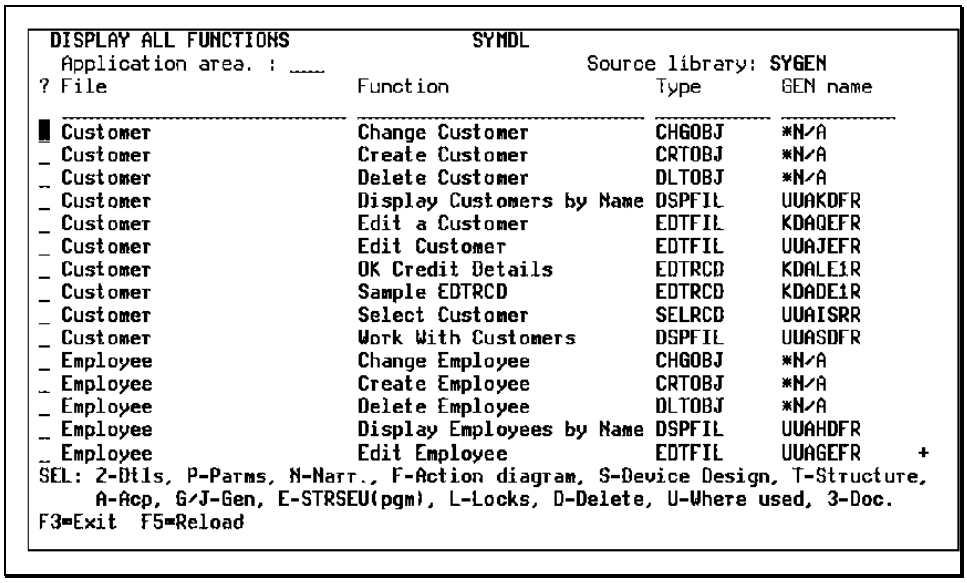
CA 2E provides certain fast path panels that allow you to display the existing functions in a design model. In this manner, you have access to the functions attached to the design model files and can perform various operations on all functions in the model from a single panel. The Display All Functions panel lists the existing functions in a design model.

Display All Functions

You access the list by pressing F17 to get to the Services Menu from which you select the Display All Functions option. You can use the positioner fields in the top portion of the display to scan for a particular file name, function name, function type, or implementation (or generation) name. You can further filter the functions on display by specifying one application area. Also, you can use any of the various command line options and function keys to

- Access the function's action diagrams, device designs, report structures, parameters, and narrative text
- Display function usage, associated access paths, and locks
- Delete and document functions

The following is an example of the Display All Functions panel.



Getting to Shipped Files and Fields

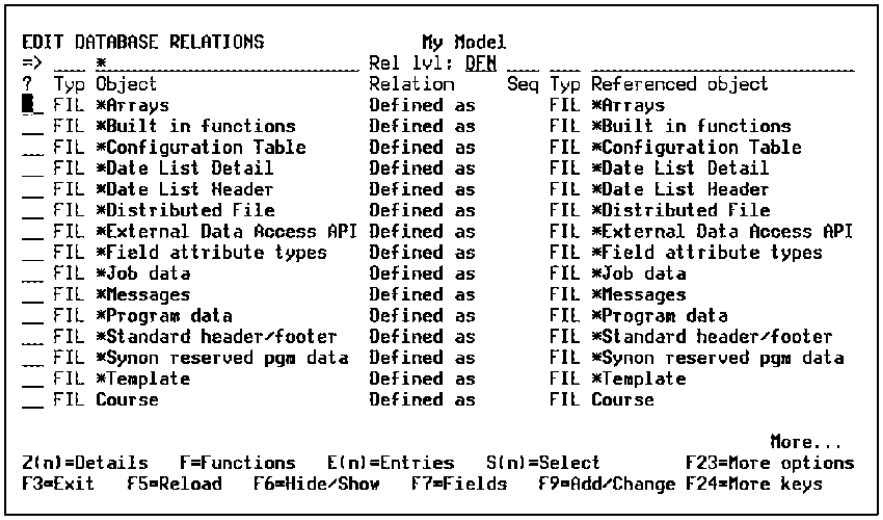
The CA 2E shipped files contain all of the default shipped data such as built-in functions, arrays, field types, job data, messages, program data, standard headers and footers, and template functions. The shipped files hold information that you use or reference in the application during the function building process.

For example, you can change the default values for fields such as return codes or confirm prompts or you can change the default functions for headers and footers.

To access the shipped files and fields

1. At the Edit Database Relations panel, type * followed by blanks on the Objects field (subfile positioner field for objects) and DFN at the relations level to show a list of files only and not the file relations. Press Enter.

   The list of shipped files appears.



2. Optionally, specify a portion of the particular file's name.

   For example by typing *St and leaving the Rel level field blank the list starts from the *Standard header/footer shipped file's relations.

Database Functions

CA 2E provides you with standard functions including the database functions described below.

Understanding Database Functions

Database functions provide the means of performing actions on the database. There are four different database functions each defining a HLL subroutine that creates, changes, deletes, or retrieves data. Database functions are implemented as part of an external standard function.

The four database functions are:

- **Create Object (CRTOBJ)**—Defines a routine to add a record to a file. It includes processing to check that the record does not already exist before writing to the database.
- **Change Object (CHGOBJ)**—Defines a routine to update a record on a file. It includes processing to check that the record already exists before updating the database record.
- **Delete Object (DLTOBJ)**—Defines a routine to delete a record from a database file. It includes processing to check that the record is still on the file before deleting it.
- **Retrieve Object (RTVOBJ)**—Defines a routine to retrieve one or more records from a database file. Processing can be specified for each record read by modifying the action diagram for the function.

A default version of the Create Change and Delete database functions is defined for all database files (REF and CPT). You must create the Retrieve Object function if you need it.

The following table includes the standard database functions.

Function	Purpose	Access Path
CRTOBJ	Add a single record	UPD, PHY
CHGOBJ	Update a single record	UPD, PHY
DLTOBJ	Delete a single record	UPD,PHY
RTVOBJ	Read a records or record	RTV,RSQ,PHY

All the CA 2E database functions have action diagrams that you can use to specify additional processing before and after the accessing the database .

Internal Database Functions and PHY Access Paths

This section contains the fields, functions, and PHY access paths

*Relative record number Field

The *Relative record number field is a 9.0 numeric field with the internal name RRN. When a program uses a physical file, a relative record number (RRN) field is assigned to that file. The *Relative record number field can be a key to access a specific record in that file, regardless of the contents of each field in that record. A different RRN field is assigned to each physical file that a program uses.

In the CHGOBJ, CRTOBJ, and DLTOBJ internal database functions built over a PHY access path, the RRN is available in the DB1 context and can be manipulated to retrieve or update a specific record.

Internal Database Functions

This section describes what happens when the following internal database functions are created over physical (PHY) access paths: Retrieve object (RTVOBJ), Change object (CHGOBJ), Delete object (DLTOBJ), and Create object (CRTOBJ).

Retrieve object (RTVOBJ)

A RTVOBJ created over a PHY access path has one *Relative record number (RRN) parameter by default:

Parameters	Usage	Role (default)	Default
*Relative record number	I	RST/POS (POS)	Y
Any other fields	Any	none	none

This *Relative record number parameter can be used as a "key" to the physical file as follows:

RRN as a Restrictor Parameter (I, B, or N) to a PHY RTVOBJ—If the RRN for a RTVOBJ function built over a PHY access path is a Restrictor parameter, only the record with RRN equal to the parameter value is read.

RRN as a Positioner Parameter (I, B, or N) to a PHY RTVOBJ—If the RRN for a RTVOBJ function built over a PHY access path is a Positioner parameter, only records with RRN greater than or equal to the parameter value are read.

RRN as an Output Parameter from a PHY RTVOBJ—If the RRN for a RTVOBJ function built over a PHY access path is an Output parameter, all records in the access path are read, starting with record 1. The RRN of the last record read from the file is passed as the parameter value.

RRN as a Neither Parameter to a PHY RTVOBJ—If the RRN for a RTVOBJ function built over a PHY access path is a Neither parameter, the RRN first used to access the file is the current value of the Neither parameter. The Neither parameter is accessible from the RTVOBJ in the PAR context. Although the parameter is initialized to 1 in the RTVOBJ, it can be changed to any numeric value in the User Exit Point USER: Initialize routine. The value following that User Exit Point is used to access the file initially.

Deleting the RRN Parameter to a PHY RTVOBJ—If the default RRN parameter is deleted, two outcomes are possible:

- If USER: Process data record User Point contains user logic, all records in the access path starting with record 1 are read.
- If USER: Process data record User Point does not contain user logic, only record 1 is read.

Note: You can add other parameters besides RRN to PHY RTVOBJ functions, but RRN must be passed first.

The following is a quick reference table for processing the *Relative record number parameter:

Usage	Role	Initialized in RTVOBJ?	Record Processing	Value Returned
I	RST	No	Single	None
I	POS	No	Single or multiple*	None
B	RST	No	Single	RRN of last record read
B	POS	No	Single or multiple*	RRN of last record read
N	RST	Neither PAR	Single	None
N	POS	Neither PAR	Single or multiple*	None
O	n/a	1	Single or multiple*	RRN of last record read
Not used	n/a	1	Single or multiple*	None

- Depends on processing in the USER: Process data record User Exit Point.

Change object (CHGOBJ)

In a normal CHGOBJ, the processing includes these steps:

1. USER: Processing before data read.
2. Load key fields to record format.
3. Access file to check if record exists.
4. USER: Processing if data record not found.
5. If record not found, send error message and quit.
6. If record locked, send error message and quit.
7. USER: Processing after data read.
8. Load non-key fields to record format.
9. USER: Processing before data update.
10. Update record.
11. If update failed, send error message and quit.
12. USER: Processing after data update.

In a CHGOBJ built over a PHY access path, the processing includes these steps:

1. Load key and non-key fields to record format.
2. USER: Processing before data update.
3. Update record.
4. If update failed, send error message and quit.
5. USER: Processing after data update.

The following notes apply to these situations:

- The pre-update file access is not generated. This is normally generated as an RPG CHAIN or as a COBOL READ statement.
- Any action diagram code in the following User Points is ignored, and no code is generated for them:
  - USER: Processing before data read
  - USER: Processing if data record not found
  - USER: Processing after data read
- A CHGOBJ created over a PHY access path can be attached only to a RTVOBJ built over the same PHY access path. This is because of the i OS requirement that a record to be changed must have been read previously.

Delete object (DLTOBJ)

In a normal DLTOBJ, the processing includes these steps:

1. USER: Processing before data update.
2. Access file to check if record still exists.
3. If record already deleted, send error message and quit.

   If record locked, send error message and quit.

5. Delete record.
6. If delete failed, send error message and quit.
7. USER: Processing after data update.

In a DLTOBJ built over a PHY access path, the processing includes these steps:

1. USER: Processing before data update.
2. Delete record.
3. If delete failed, send error message and quit.
4. USER: Processing after data update.

The following notes apply to these situations:

- A DLTOBJ created over a PHY access path is created with no parameters. You must ensure that the record to be deleted was read in a RTVOBJ built over the same PHY access path. Thus, the DLTOBJ should be inserted only in the USER: Process data record User Exit Point in the RTVOBJ.
- The pre-delete file access is removed. This is normally generated as an RPG CHAIN or as a COBOL READ statement.
- Any code in the USER: Processing if data record already exists User Point is ignored.
- A DLTOBJ built over a PHY access path can be attached only to a RTVOBJ built over the same PHY access path.

Create object (CRTOBJ)

In a normal CRTOBJ, the processing includes these steps:

1. Load parameters to record format.
2. USER: Processing before data update.
3. Access file to check if record already exists.
4. USER: Processing if data record already exists.
5. If record already exists, send error message and quit.
6. Write record.
7. USER: Processing if data update error.
8. If write failed, send error message and quit.
9. USER: Processing after data update

In a CRTOBJ built over a PHY access path, the processing includes these steps:

1. Load parameters to record format.
2. USER: Processing before data update.
3. Write record.
4. USER: Processing if data update error.
5. If write failed, send error message and quit.
6. USER: Processing after data update.

The following notes apply to these situations:

- The pre-create file access is removed.
- Any code in the USER: Processing if data record already exists User Point is ignored.
- A CRTOBJ built over a PHY access path cannot be used in any function that also contains a RTVOBJ built over the same PHY access path. This is because the file definition requirements of a PHY access path used for CRTOBJ are different from those used for CHGOBJ, DLTOBJ, or RTVOBJ. However, the Action Diagram Editor registers an error only if you attempt to attach the CRTOBJ to a RTVOBJ directly. If you attach a CRTOBJ to, for example, an EXCEXTFUN that also contains a PHY RTVOBJ, the Editor does not register an error, but the function compilation will fail.

Using Functions Built Over PHY Access Paths

This is a quick reference table with information about functions built over PHY access paths:

Database Function	Attaching to:	Allowed by Compiler?	Action Diagram Editor Error?
Retrieve object	CRTOBJ over same PHY access path	No	Yes
Retrieve object	Other function	Yes	n/a
Change object	RTVOBJ over same PHY access path	Yes	n/a
Change object	Other function	No	Yes
Delete object	RTVOBJ	Yes	n/a
Delete object	Other function	No	Yes
Create object	RTVOBJ over same PHY access path	No	Yes
Create object	Other function containing RTVOBJ	No	No
Create object	Other function	Yes	n/a

Consider the following points when using functions built over PHY access paths:

- Because some error checking has been removed from these functions, the application designer must ensure that applications using these functions do not run at the same time as other functions that use these files. Otherwise, locks may be placed on records that these functions need to read.
- CHGOBJ and DLTOBJ functions built over a PHY access path can be used only in a RTVOBJ built over the same PHY access path. This ensures that the record to be changed or deleted has just been read in the RTVOBJ.
- A CRTOBJ function built over a PHY access path can be used only in a function that does not contain a RTVOBJ, CHGOBJ, or DLTOBJ built over the same PHY access path. We suggest that you create an Execute external function (EXCEXTFUN) with the same parameters as the CRTOBJ, include only the CRTOBJ in that function, and access the PHY CRTOBJ by using that function.
- Although the generators create code differently for the RTVOBJ, CHGOBJ, DLTOBJ, and CRTOBJ functions, the action diagram for each function does not change. This may cause confusion, because User Points are visible in the action diagram and statements can be entered in them, but those User Points may not be generated.

Array Processing

To add, delete, modify, or retrieve entries in a particular array over which they are defined, use the following database functions:

- Create Object (CRTOBJ)
- Delete Object (DLTOBJ)
- Change Object (CHGOBJ)
- Retrieve Object (RTVOBJ)

A DLTOBJ with no parameters clears an array.

Although arrays are not implemented as database files, CA 2E allows you to use the same techniques as database files when working with arrays.

Note: You must define a key for an array even if the array holds a single element.

Device Functions

In addition to the database functions, CA 2E also provides standard device functions as follows.

Understanding Device Functions

Device functions are interactive panels or reports. Panel device functions present the interactive user interface between the end user and the application. Report device functions provide a method of defining a written presentation of data. All device functions, with the exception of PRTOBJ, are implemented as external functions. PRTOBJ is an internal function.

Defining Device Functions

CA 2E provides comprehensive interactive design facilities that allow you to specify a panel or report layout. CA 2E interactive device design editor allows you to define field attributes, positioning, conditioning, user function keys, and panel or report literals for interactive display or written presentation.

You access this interactive editor from the Edit Functions, Edit Function Devices, or the Display All Functions panels.

The device standard header device functions are:

- Define Screen Format (DFNSCRFMT)—This function allows you to define a standard screen header and footer for use by other functions that have screen designs attached to them
- Define Report Format (DFNRPTFMT)—This function allows you to define a standard report header and footer for your Print File report functions

The single-record device functions are:

- **Prompt Record (PMTRCD)**—Defines a program to prompt for a list of fields defined by a specified access path. The validated values can be passed to any other function.
- **Display Record (DSPRCD)**—Defines a program to display a single record from a specified database file. If no key is supplied, a key panel prompts for a key.
- **Display Record 2 panels (DSPRCD2)**—Defines a program that is identical to the DSPRCD function, except that it allows the database record details to extend to two separate display device pages.
- **Display Record 3 panels (DSPRCD3)**—Defines a program that is identical to the DSPRCD function, except that it allows the database record details to extend to three separate display device pages.
- **Edit Record (EDTRCD)**—Defines a program to maintain (add, change, and delete) records on a specified file, one at a time. If no key is supplied, a key panel prompts for a key.
- **Edit Record 2 panels (EDTRCD2)**—Is identical to the Edit Record function, except that it allows the record details to extend to two separate display pages.
- **Edit Record 3 panels (EDTRCD3)**—Is identical to the Edit Record function, except that it allows the record details to extend to three separate display pages.

The multiple-record device functions are:

- **Display File (DSPFIL)**—Defines a program to display the records from a specified file, many at a time, using a subfile. The subfile is loaded a page at a time when you press Rollup or F8.
- **Select Record (SELRCD)**—Defines a program that displays the records from a specified file, many at a time, using a subfile. The program allows you to select one of the records. The selected record is returned to the calling program. This function is called from a function that requested a selection list.
- **Edit File (EDTFIL)**—Defines a program to maintain the records on a specified file, many at a time, using a subfile. The subfile is loaded a page at a time when you press Rollup or F8**.**

The single- and multiple-record device functions are:

- **Display Transaction (DSPTRN)**—Defines a program to display the records from a specified pair of database files. The pair must be connected by an Owned by or Refers to relation.
- **Edit Transaction (EDTTRN)**—Defines a program to maintain the records on a specified pair of header and detail files. The pair must be connected by an Owned by or Refers to relation.

The printer device functions are:

- **Print File (PRTFIL)**—Defines a program to print records from a specified access path.
- **Print Object (PRTOBJ)**—Defines a particular report fragment which prints the records from a specified access path at any point within a Print File function. Print Object functions can be embedded within other Print Object functions.

Device Functions'Standard Features

All HLL programs that implement device functions use standard techniques for each of the following aspects of interactive programs.

Standard Features—User Interface

- **Diagnostic messages**—If an error is detected in a CA 2E generated program, a message is sent to the program's message queue. All interactive programs have a message subfile to show the pending messages on the program's message queue. This message handling technique makes full use of the sophisticated message handling capabilities of i OS, allowing both second level text and substitution variables. It also ensures that applications can be translated easily into other national languages.
- **Highlighting of errors**—Any field found to be in error is highlighted in reverse image. The cursor is positioned at the first of these fields.
- **On-line Help text**—All the interactive programs generated by CA 2E include processing to call a program to display Help text when the Help key or F1 is pressed.
- **Print key**—The print key is enabled to allow panel prints. The name of the print key spool file can be controlled with the YPKYVNM model value.
- **Selection columns**—Subfiles that allow selection of individual items always have the selection column on the left.
- **Function key usage**—Function key usage is standardized to follow CUA standards: for example, F3 is exit.

  For more information on function key usage, see Function Key Defaults in the chapter "Setting Default Options for Your Functions."
- **Positioning facilities**—When appropriate, programs that use subfiles have a positioning field on the subfile control record that you can use to control which records are shown in the subfile.

Standard Features—Processing Techniques

- **Single Page Subfile (SFL) load up**—Programs that use SFLs only load the SFL on a demand basis. Normally, this means only when the Rollup key or F8 is pressed. This makes their performance more efficient. However, the device function types that need to read all of a restricted number of records (namely the EDTTRN and DSPTRN functions) reads more than a page of records at a time if appropriate.
- **Concurrency checking and record locking**—Programs that update the database do not generally hold a lock on the database while the changes to the database are being entered and validated; that is, between reading an existing record and updating it. Instead, they include processing at the point of update to check that records were not altered by other users since the record was first accessed by the updating program. This approach prevents locking out any concurrent users or batch processes who or which may also need to update the file.
- **Overflow handling**—CA 2E generated report functions include exception handling to cope with page overflow. You can specify whether headings are reprinted or not.

Device Function Program Modes

Each of the programs specified by CA 2E standard device functions operate in one or more modes, depending on the function type. Program modes give the user a simple way of controlling program behavior.

The following table shows the program modes by function type.

Function Type	*ADD	*CHANGE	*SELECT	*DISPLAY	*ENTER
PMTRCD		-	-	-	Y
DSPRCD1,2,3	-	-	-	Y	-
DSPFIL	-	-	-	Y	-
EDTRCD1,2,3	Y	Y	-	-	-
EDTFIL	Y	Y	-	-	-
SELRCD	-	-	Y	-	-
DSPTRN	-	-	-	Y	-
EDTTRN	Y	Y	-	-	-

Note: Program modes do not apply to report functions.

Classification of Standard Functions by Type

The following table lists the standard function types.

Function Type	Abbreviation	Class	Imp	Dev	Action Diagram	Params	Function Option
Retrieve Object	RTVOBJ	dbf	int	-	Y	Y	Y
Change Object	CHGOBJ	dbf	int	-	Y	Y	Y
Create Object	CRTOBJ	dbf	int	-	Y	Y	Y
Delete Object	DLTOBJ	dbf	int	-	Y	Y	Y
Define Screen Header	DFNSCRDSN	dfn scr	int	-	-	-	Y
Define Report Header	DFNRPTDSN	dfn rpt	int	-	-	-	Y
Prompt and Validate	PMTRCD	dev scr	ext	Y	Y	O	Y
Display Record	DSPRCD	dev scr	ext	Y	Y	O	Y

Function Type	Abbreviation	Class	Imp	Dev	Action Diagram	Params	Function Option
Display Record (2 panels)	DSPRCD2	dev scr	ext	Y	Y	O	Y
Display Record (3 panels)	DSPRCD3	dev scr	ext	Y	Y	O	Y
Edit Record	EDTRCD	dev scr	ext	Y	Y	O	Y
Edit Record (2 panels)	EDTRCD2	dev scr	ext	Y	Y	O	Y
Edit Record (3 panels)	EDTRCD3	dev scr	ext	Y	Y	O	Y
Display File	DSPFIL	dev scr	ext	Y	Y	O	Y
Select Record	SELRCD	dev scr	ext	Y	Y	O	Y
Edit File	EDTFIL	dev scr	ext	Y	Y	O	Y
Display Transaction	DSPTRN	dev scr	ext	Y	Y	O	Y
Edit Transaction	EDTTRN	dev scr	ext	Y	Y	O	Y
Print File	PRTFIL	dev rpt	ext	-	Y	O	Y
Print Object	PRTOBJ	dev rpt	int	Y	Y	O	Y
Execute Internal Funct.	EXCINTFUN	usr	int	-	Y	O	Y
Execute External Funct.	EXCEXTFUN	usr	ext	-	Y	O	Y
Execute User Program	EXCUSRPGM	usr	ext	-	-	O	Y
Execute User Source	EXCUSRSRC	usr	int	-	-	O	-
dbf = database file dev = device dfn = define	ext = external int = internal O = Optional				rpt = report scr =screen usr = use		

User Functions

CA 2E provides you with standard user functions as described in the following sections.

Understanding User Functions

User functions provide the means of implementing additional user processing within an existing function or as an independent implementation used in conjunction with an existing function. There are four basic user functions: Execute External Function, Execute Internal Function, Execute User Program, and Execute User Source.

Function	Action Diagram	Implementation
EXCEXTFUN	Yes	External
EXCINTFUN	Yes	Internal
EXCUSRPGM	No	External
EXCUSRSRC	No	Internal

Defining Free-Form Functions

Free-form user functions provide the means of specifying actions that can be used within a function or called from a function to perform a series of procedures. These functions do not conform to any predefined structure and the contexts of these functions are entirely composed of actions. You define these function types at the Edit Functions panel. The processing logic for these functions is defined with the Action Diagram Editor.

The free-form functions are:

- **Execute Internal Function (EXTINTFUN)**—This function allows you to specify a section of an action diagram for repeated use in other functions.
- **Execute External Function (EXCEXTFUN)**—This function allows you to specify a HLL program using an action diagram.

Defining User-Coded Functions

User coded functions are functions that are user-written in a HLL. They can be called from another function or embedded within a function.

The user-written coded functions are:

- **Execute User Program (EXCUSRPGM)**—This function allows you to describe the interface to a user written HLL program so that it can be referenced by functions. Parameters can be specified on the call.
- **Execute User Source (EXCUSRSRC)**—This function specifies either:
  - User-written HLL code to perform an arbitrary function that is to be included within the source generated by CA 2E for an HLL program.
  - Device language statements, for example, DDS that can be applied to a device function to customize the associated device design.

You define these function types at the Edit Functions panel. The user-coded functions are called or referenced by any function. However, they do not have an associated action diagram. You can edit the source directly from within CA 2E.

An EXCUSRPGM function generally is an existing program that you integrate into your application. This process typically requires you to rename the default name for the function to the name of the existing user program. DDS names must match for EXCUSRPGM or source copied into functions.

EXCUSRSRC function types must be of the same HLL source type as that of any functions that call them.

Messages

CA 2E provides you with standard message functions. They are described below.

Understanding Messages

An i OS message file is an i OS object that contains individual message descriptions. A message description is a unit within the message file that contains specific information. The message description includes the message identifier, the message text, and other details about the message. You specify substitution variables that allow data to be inserted within the text when the message is used.

The message functions allow the user to

- Define messages of varying types
- Specify different message files to which the message is attached
- Specify substitution variable parameters
- Change message identifiers

Basic Properties of Messages

CA 2E provides default system names for messages and provides the means to override the message file names and message identifiers. CA 2E provides six message types: completion, error, execution, information, retrieval, and status. Parameters can be defined for message functions. Parameters correspond to fields or files.

CA 2E provides default messages that correspond to default logic processing inherent in CA 2E external functions. These messages include default existence and not found messages created for all files.

The message functions are:

- **Send Error Message (SNDERRMSG)**—This function specifies that an error message be sent to a calling function. Normally, this function is used to provide diagnostic messages arising from user validation.
- **Send Information Message (SNDINFMSG)**—This function specifies that an informational message be sent to the message queue of a calling program.
- **Send Completion Message (SNDCMPMSG)**—This function specifies that a completion message be sent to the function that called a standard function. Typically, completion messages are used to indicate that a process completed successfully.
- **Send Status Message (SNDSTSMSG)**—This function specifies that a status message be sent to a calling function. Normally, this function is used to provide information about the progress of a long-running process.
- **Retrieve Message (RTVMSG)**—This function specifies that message text be retrieved from the message file into a function.
- **Execute Message (EXCMSG)**—This function specifies that a request message be executed. The request can be any CL command.

Defining Message Functions

You define your message functions using the following instructions.

Specifying Message Functions Details

Message functions are defined at the Edit Message Functions panel.

1. At the Edit Database Relations panel, type *****M in the Object field and press Enter to get to the message subfile.

   The *MESSAGES file appears.

2. Type **F** next to a relation for the selected file.

   The Edit Message Functions panel appears.

3. Press **F9** to define a new message.

4. Go to a blank subfile line on the Edit Message Functions panel.

5. Enter the message function name and message type from one of the available option types described previously. If you are uncertain of the type of message, type **?** in the Type field to display a list of valid values.

Note: You can define and modify messages while editing an action diagram.

Specifying Parameters for Messages

A parameter is used within the text portion of a message. During generation, the parameter's value displays. Parameters can be specified for a message function using the following instructions:

1. Use the previous instructions to get to the Edit Message Functions panel.
2. Type **P** next to the selected message function.

   The Edit Function Parameters panel appears.
3. Define the parameter.

Note: When the data type of a parameter allows value mapping, such as all date and time fields, the parameter is generally converted to its external format before the message is sent. However, due to limitations within i OS, the parameter data for the TS# data type is passed in its internal format, namely, YYYY-MM-DD-HH.MM.SS.NNNNNN.

A parameter can be defined for a message function to allow substitution of the parameter's value into the text portion of the message identifier.

For example, to insert a field's value in an error message when the credit limit is exceeded for a customer, enter the following:

```
Credit limit exceeded for &1.
```

The parameter value &1 is inserted into the message text at execution time. You must then define &1 as an input parameter value to the message function. If this is an error message, it also causes the field associated with the parameter &1 to display using the error condition display attribute for the field. By default, this is reverse image.

Specifying Second-Level Message Text

Second-level text defines a full panel of information that you can choose to display for any message that is issued. It is also used to define the text of messages to executed on a platform-by-platform basis. To specify second-level message text:

1. Use the previous instructions to get to the Edit Message Functions panel.
2. Type **Z** next to the selected message function.

   The Edit Message Function Details panel appears.
3. Press **F7**.

   The Edit Second Level Message Text panel appears.
4. Specify the second-level message text.

Function Fields

CA 2E provides function fields. They are described in the following sections.

Understanding Function Fields

Function fields are special types of fields that you can use in device designs and action diagrams. The attributes of a function field are typically based on other fields. In addition, to specify the field definition you can optionally specify processing for a particular function field based on the function field usage.

Basic Properties of Function Fields

There are six different types or usages of function fields. The following four usages provide standard field level functions:

- Sum (SUM)
- Count (CNT)
- Maximum (MAX)
- Minimum (MIN)

The other two fields enable you to define your own function fields, either with or without a user-specified calculation to derive the field. These function field usage types are:

- Derived (DRV)
- User (USR)

Function field parameters specify which field values are passed into the function at execution time and, inversely, which field is returned from the function as the result field.

Derived (DRV) function fields must have one output parameter and can have many input parameters.

Maximum (MAX), Minimum (MIN), Count (CNT) and Sum (SUM) function fields have only one output parameter (the field itself) and only one input parameter that defines a field on which the calculation is based.

USR usage function fields have no associated parameters. These fields are typically used as work fields in an action diagram.

DRV usage function fields have associated action diagrams. A free-form action diagram shell (such as for EXCINTFUN) is associated with the derived function field to specify processing steps.

Design Considerations

Function fields can be pulled into a panel display or a report. The function fields appear on the device design. However, the special characteristics inherent in each function field type allow you to specify unique processing.

For example, a user could specify a SUM function field to sum a computed total for all of the detail lines on an EDTTRN function called Edit Orders display. The SUM field is used to compute a value from an occurrence of a field in a detail format, with the result placed in the summation field in a header format.

Note: The totaling function fields, MIN, MAX, SUM, and CNT are only valid for header or detail display functions such as Display Transaction and Edit Transaction as well as print functions (Print Object and Print File).

Defining Function Fields

Function fields are defined in the same way that database fields are defined by using the Define Objects panel. You can access this panel by pressing F10 on the Edit Database Relations panel or by pressing F10 on the Display All Fields panel.

Because you can access the Display All Fields panel while editing an action diagram or device design, you can define function fields while performing other activities.

For more information on defining function fields, refer to this module, in the chapter, "Modifying Device Designs."

Function fields that require the specification of parameters are DRV, SUM, MIN, MAX, and CNT. The function field that requires an action diagram is DRV.

For more information on function fields, refer to this module, in the chapter, "Modifying Action Diagrams."

Function Types, Message Types, and Function Fields

The CA 2E function types, message types, and function fields are listed in alphabetical order on the following pages with a detailed description of each.

For more information on the specific user points for these function types, see the Understanding User Points topic in the chapter, "Modifying Action Diagrams."

Database Function

CHGOBJ The Change Object (CHGOBJ) function defines a routine to update a record in a file. The CHGOBJ function includes the processing to check that the record exists before updating the database record.

There are no device files associated with the CHGOBJ function. However, it does have action diagram user points. This function must be attached to an update access path.

Note: For more information on PHY, see the section Internal Database Functions and PHY Access Paths.

The default CHGOBJ function is used to update all fields in the database record. If you want to change only a subset of the fields on the file, you must define a new CHGOBJ function specifying each field to be excluded as a Neither parameter or define a CHGOBJ based on a different access path containing only those fields.

A CHGOBJ function is provided for every database file and is based on the primary update access path. It is inserted in the Change DBF Record user point in edit functions that update changed records.

Note: When a CHGOBJ is inserted in the Change DBF Record user point, code is generated to check if the record was changed (by another user) before the record is updated. However, if a CHGOBJ is inserted in any other user point of any function type, this checking is not generated.

All fields on the access path must be provided as Neither, Input, or Both parameters and cannot be dropped.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
All fields from access path	I	-	Y	R
Any other fields	Any	-	-	O

The following table shows the function options available.

Option	Default Value	Other Values
Null Update Suppression	M(YNLLUPD)	N, Y, A
Share Subroutine	M(YSHRSBR)	N, Y

Null Update Suppression Logic

The null update suppression logic generated in CHGOBJ functions determines whether to update the database record as shown in the following steps:

1. Before the After Data Read user point, CHGOBJ saves an image of the original data record and initializes the *Record data changed PGM context field to ‘ ‘.
2. CHGOBJ performs the following checks.
   - Compares the saved image and the current image of the record to determine whether the data record has changed
   - Checks whether logic in the preceding user point explicitly set the *Record data changed PGM context field to *NO in order to force suppression of the data update

   If the images differ *and* the *Record data changed field is *not* *NO, CHGOBJ sets the *Record data changed field to *YES.

   Note: Where and how often the previous checks are done within the CHGOBJ depends on whether YNLLUPD is *AFTREAD or *YES. If *YES, check is done *both* after the After Data Read and after the Before Data Update user points. If *AFTREAD, the check is done only after the After Data Read user point.
3. Before updating the database record, CHGOBJ checks the *Record data changed PGM context field. If it is *YES, the database record is updated, otherwise the record is released.

For more information about:

- YNLLUPD values, see the CHGMDLVAL command in the *Command Reference*.
- The *Record data changed field and an example, see, Understanding Contexts, PGM in the chapter "Modifying Action Diagrams"

It is possible to change the primary key of a file using CHGOBJ. However, this is only valid for RPG/DDS and this generally violates relational database principles. Changing the primary key should be performed using DLTOBJ function followed by a CRTOBJ function. In COBOL or SQL, a primary key change must be performed in this way.

For a given based-on access path, if you want to update all database fields in some functions but only a subset of fields in other functions, create a second CHGOBJ function by copying the default CHGOBJ function. On the second CHGOBJ, specify the fields you do not want to update as Neither (N) parameters. Only fields specified as Input (I) parameters are updated in the database record. Use the second CHGOBJ instead of the default CHGOBJ in the functions where you want to update the subset of the database fields.

For more information about:

- How to use a DLTOBJ function, see DLTOBJ later in this chapter

- The user points for the CHGOBJ function, see the chapter "Modifying Action Diagrams"

CNT Function Field

The Count (CNT) function field is a field usage type used within certain functions (EDTTRN, DSPTRN, PRTFIL, and PRTOBJ) to define a count of a set of records. The CNT function field must be based on one of the fields in the record format.

In order for CNT to determine which records to count, you must point it to a record on the device. To do this, CNT must reference one of the fields in the record. The actual field selected and the values in that field do not affect the result of the CNT function. The CNT field itself must be a numeric field.

CNT function fields always have two parameters:

- **A result parameter**—This is the actual field itself containing the results of a summation. You must place the field on a totaling format of the function that uses the CNT function field.
- **An input parameter**—This represents a summation of the number of instances or occurrences of the field being passed. Your input parameter must be on a field on the detail or subfile record format of the function using the CNT function field.

Note: If you reference this function field to another field, that field defaults to the input parameter of the CNT function field.

Examples of Count fields:

- Number of employees in a company
- Number of order lines in an order

CRTOBJ Database Function

The Create Object (CRTOBJ) function defines a routine to add a record to a database file. The CRTOBJ function includes the processing to check that the record does not already exist prior to being written to the database. There are no device files associated with the CRTOBJ function; however, it does have action diagram user points.

A CRTOBJ function is provided by default for every database file. All CRTOBJ functions are attached to an Update (UPD) access path, or can be attached to a Physical (PHY) path.

Note: For more information on PHY, see the section Internal Database Functions and PHY Access Paths

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
All fields from access path	I	-	Y	R
Any other fields	Any	-	-	O

The following table shows the function options available.

Option	Default Value	Other Values
Share Subroutine	M(YSHRSBR)	N,Y

All fields from the UPD access path must be declared as parameters to the CRTOBJ function. To exclude certain fields from being written, you should specify them as Neither parameters. These fields are not automatically initialized. If you use Neither parameters, you should initialize the fields with blank or zero.

If the UPD access path to which the CRTOBJ function is attached does not contain all of the fields in the based-on file, the missing fields are set to blank or zero. You can change this by specifying a value on the Default Condition field of the Edit Field Details panel.

For more information on user points, see the chapter, "Modifying Action Diagrams."

DFNSCRFMT Device Function

The Define Screen Format (DFNSCRFMT) function defines a standard panel header and footer for use by other functions that have panel designs attached to them.

There are three default Define Screen Format functions shipped as standard header/footer formats for the device function panel design:

- *STD SCREEN HEADINGS (CUA) function follows the SAA CUA Entry Model standards
- *STD CUA WINDOW function follows CUA standards for window panels.
- *STD CUA ACTION BAR function follows CUA standards for action bar panels

You can modify these shipped versions as well as add your own DFNSCRFMT functions for use in specific function panel designs. You can use the Edit Function Options panel of any panel function to change the DFNSCRFMT function used for that particular function.

When a model is created, the defaults depend on the value given to the DSNSTD parameter on the Create Model Library command, YCRTMDLLIB.

Use the function options for the DFNSCRFMT functions to set the defaults to be used by newly created device functions.

Attach the DFNSCRFMT function to the physical (PHY) file access path of the *Standard header/footer file. This function type does not allow parameters. It does not have an action diagram.

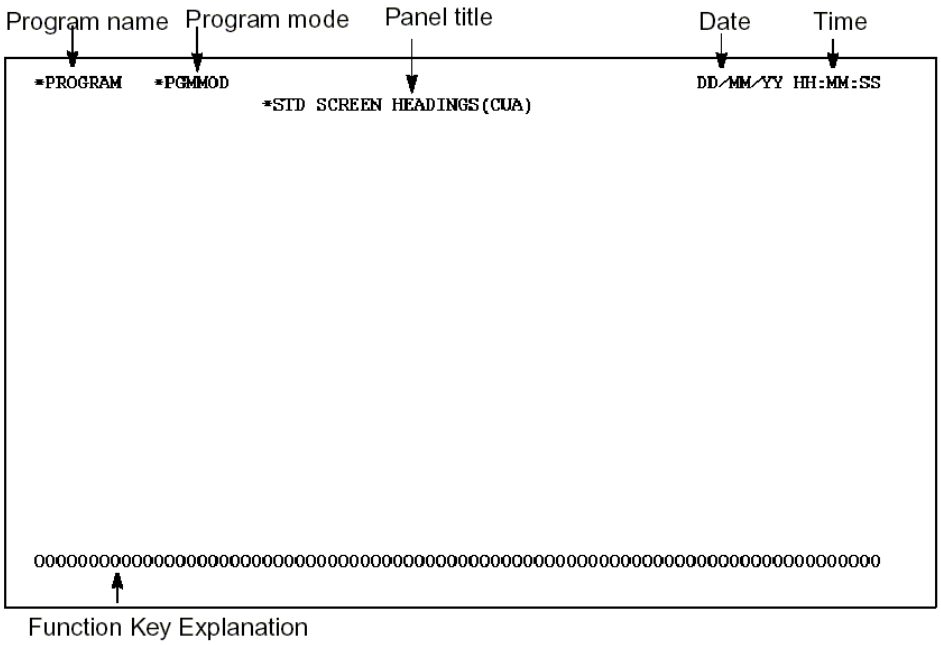
If you define an additional DFNSCRFMT function, all the fields from the *Standard header/footer file are available on both formats of the function. You can rearrange or suppress these fields.

By default, the header and footer formats are as follows:

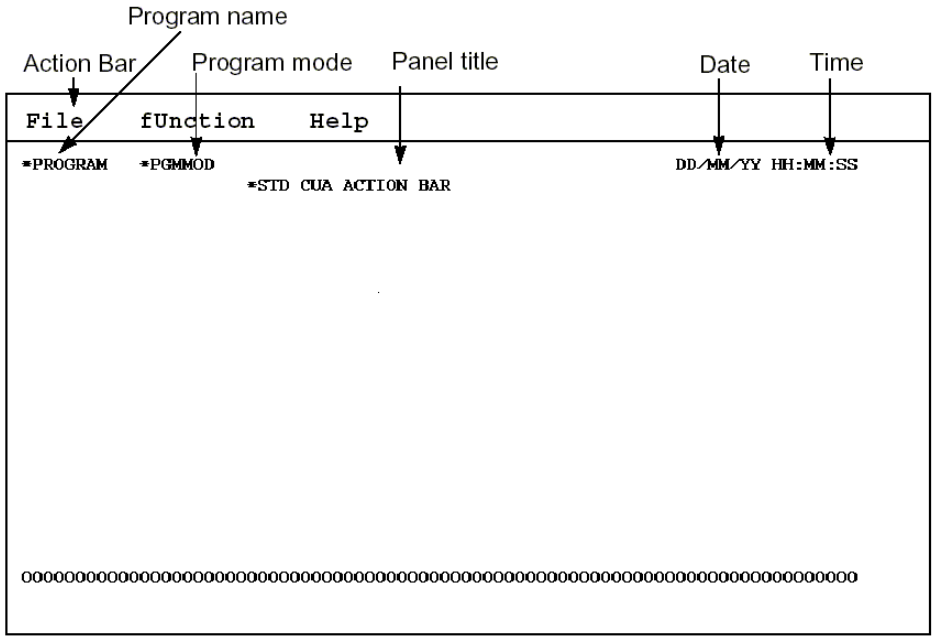
- For CUA Entry panels, the title is at the top and the command area at the bottom; for CUA Text, the panel includes an action bar. You can change the location of the command area using the Edit Screen Format Details panel.
- The fields that can be included in the design of a DFNSCRFMT function are included in the CA 2E shipped file, *Standard header/footer.

Header and footer formats have instruction lines that are hidden by default.

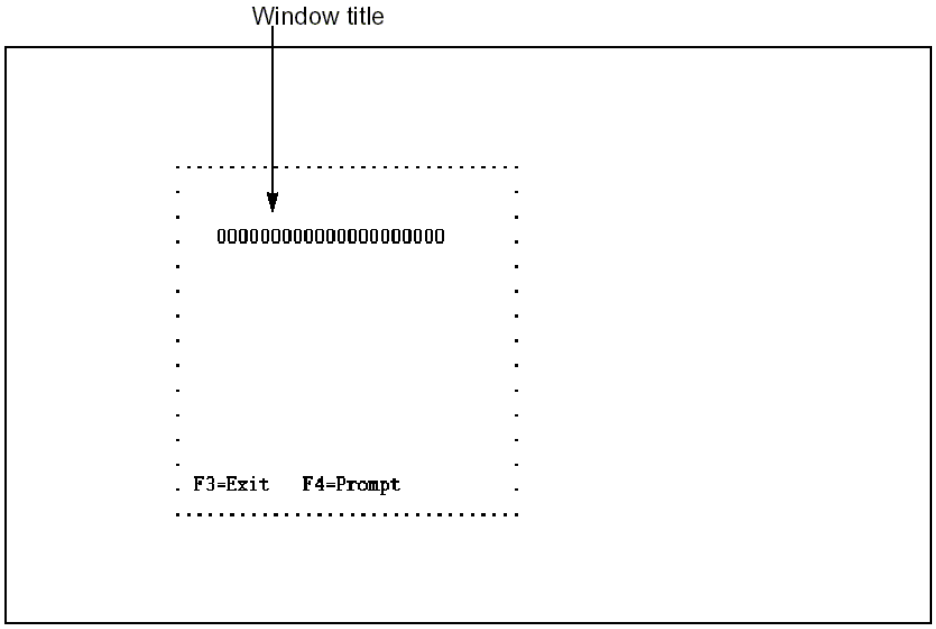
The following is an example of a CUA Entry Standard panel.



The following is an example of a CUA Text Standard panel.



The following is an example of a CUA Text Standard window.



For more information on the function options available, see Identifying Standard Header/Footer Function Options in the chapter, "Modifying Function Options."

DFNRPTFMT Device Function

The Define Report Format (DFNRPTFMT) function defines a standard report header and footer for your Print File report functions.

A default DFNRPTFMT function is shipped with CA 2E to define standard header/footer formats for device function report designs. You can modify the shipped version or add your own DFNRPTFMT functions for use in specific function report designs. Any new report device function created by CA 2E uses the shipped DFNRPTFMT function by default, unless you nominate a different default. To use your own DFNRPTFMT, you can change the DFNRPTFMT for any report function using the Edit Function Options panel.

The fields that can be included in the design of a DFNRPTFMT function are included in the CA 2E shipped file called *Standard header/footer.

Attach the DFNRPTFMT function to the physical (PHY) file access path of the *Standard header/footer file.

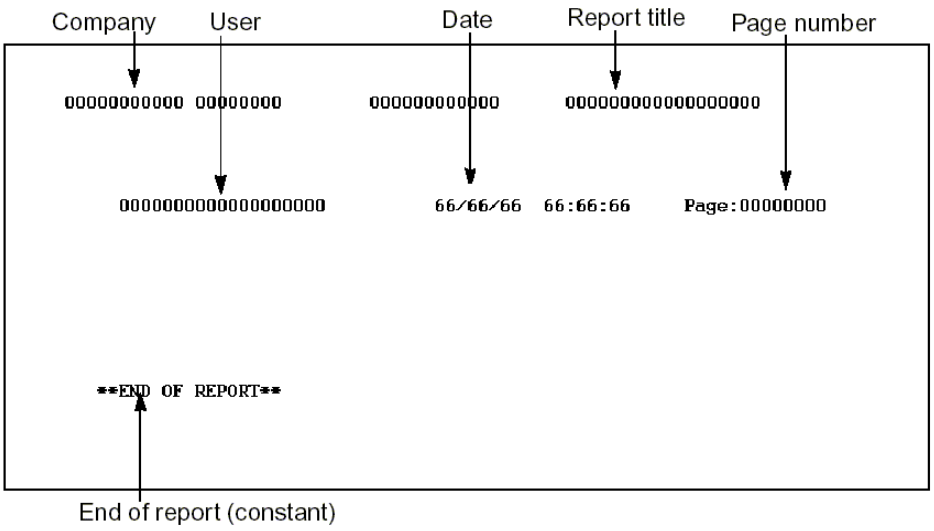
This function type does not allow parameters. It does not have an action diagram.

By default, the header format for the report starts on line one. This value can be changed using the Edit Device Format Details panel.

The shipped standard report page header is for a report that is 132 characters wide. To define a DFNRPTFMT function with a different report width, specify a different value for the PAGESIZE parameter on the overrides to the i OS Create Print File command (CRTPRTF) for the function. You do this using the overrides prompt available from the Edit Function Details panel (F19 function key).

Use the function options for the DFNRPTFMT function to set the default header to be used by newly created report functions.

The fields of the header are shown in this example in two lines (in practice, the header extends beyond the limit of one panel).



Note: Any changes that you want to make to a report header and footer of a report function must be made by modifying the DFNRPTFMT function associated with the function. The header fields of report formats shown on the report function itself are protected.

For more information on the function options available, see Identifying Standard Header/Footer Function Options in the chapter, "Modifying Function Options.".

DLTOBJ Database Function

The Delete Object (DLTOBJ) function defines a routine to delete a record from a database file. The DLTOBJ function includes processing to check that the record is still on the file before deleting it. You can add processing to a DLTOBJ function such as processing that performs a cascade delete of any associated records.

The DLTOBJ function does not have device files or function options. However, it has action diagram user points.

A DLTOBJ function is provided for all files. By default, it is normally inserted into the Delete DBF Record user point in all edit functions.

Note: When a DLTOBJ is inserted in the Delete DBF Record user point, code is generated to check if the record has been changed (by another user) before the record is deleted. However, if a DLTOBJ is inserted in any other user point in any function type, this checking is not generated.

The DLTOBJ function must be attached to an Update (UPD) access path, or can be attached to a Physical (PHY) access path. The primary key(s) of the UPD access path must be declared to the Delete Object function; other parameters can also be added.

Note: For more information on PHY, see the section Internal Database Functions and PHY Access Paths

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Key fields from access path	I	-	Y	R
Other fields	Any	-	-	O

The following table shows the function options available.

Option	Default Value	Other Values
Share Subroutine	M(YSHRSBR)	N, Y

You can add your own checks to the action diagram of the DLTOBJ function to ensure that no record is deleted that is referenced elsewhere in the database, ensuring referential integrity. You would use a Retrieve Object function to check for the existence of referenced records.

For example, you can allow Customers to be deleted from the Customer file but only if the Customer is not referenced in an order on the Order file. You could modify the DLTOBJ function for the Customer file to include this check by calling a Retrieve Object function on the Order file. If an Order is found that references the Customer, the condition Error is moved to the program's *Return code and tested on return to the Delete Object function:

```
>  USER: Processing before DBF update
:—
: Retrieve object — Order *                                          <<<
: .-CASE<<<                                                          <<<
: |-PGM. *Return code is Error <<<                                   <<<
: |,—QUIT<<<                                                        <<<
: ,-ENDCASE<<<                                                       <<<
'—
```

Array DLTOBJ

You can use a DLTOBJ to delete one element of an array or to clear the contents of an array. To clear the entire array, define a DLTOBJ over the array and delete all parameter entries from the DLTOBJ. When this special type of DLTOBJ executes, it clears the associated array.

DRV Function Field

The Derived (DRV) function field usage is a special field used within functions to perform a user-defined action to derive the result field. An empty action diagram is initially associated with the function field.

A DRV field always has one output parameter: the derived field itself. You can specify as many additional input parameters as required.

Example of a Derived Function Field

An example of a DRV field is a total value field that contains the result of the calculation of the Quantity multiplied by Price fields. Other examples include retrieval of data from an access path.

For example, rather than having a virtual field in a file, you could use a derived function field that includes a RTVOBJ to read the file and return a value to the field.

Example of a Compound Condition with Derived Fields

A DRV function field can be used to encapsulate a compound condition. The result would be a true/false condition. This can be used in an action diagram or to condition a device field. Similarly, DRV function fields can be used to encapsulate a compute expression.

- Derived fields are equivalent to a function call that returns a single variable.
- Derived fields can be used in any function type and are not restricted to device functions.

For more information on function fields, see Function Field presented earlier in this chapter.

DSPFIL Device Function

The Display File (DSPFIL) function defines a program to display a list of records from a specified file using a subfile. The subfile is loaded one page at a time when the scroll keys are pressed.

The DSPFIL function also allows you to select specific field values within a file by using fields in the subfile control area of the panel. For each field on the subfile record, an associated input-capable field is, by default, present on the subfile control area of the panel design.

Effects of Parameters

Restrictor Parameters are only for key fields. Specifying restricted key (RST) parameters to the DSPFIL function restrict the records that are written to the subfile for display. Only records from the based-on access path with key values that match exactly these values appear. Restrictor parameters are output only.

Positioner Parameters are only for key fields. The effect of positioner key fields is that only records from the based-on access path with key values greater than or equal to the specified positioner key values appear in the subfile. Any key field that is defined to the DSPFIL function as an Input/Mapped parameter acts as a positioner for the subfile display.

Selector Parameters are only for non-key fields. The effect of selector fields is that only records from the based-on access path with data values that match the precise value specified on the selector field display. You can define the specific criterion for the selector field on the Edit Screen Entry Details panel.

The choices of selector criteria are:

- Equal to (EQ)
- Not equal to (NE)
- Less than (LT)
- Less than or equal to (LE)
- Greater than or equal to (GE)
- Contains (CT)
- Greater than (GT)

Any non-key field that is defined as an Input/Mapped parameter acts as a selector for the subfile display. You should drop, not hide, any fields from the control format that you do not want to use as selectors. You should drop all positioners and selectors from the device that you do not require for your function or additional processing can occur that is not required.

A DSPFIL function can be attached to a Retrieval (RTV), Resequence (RSQ), or Query (QRY) access path. The access path determines which records are displayed by this function. Further selection of records can be made by specifying whether a record is selected in the appropriate point in the action diagram. The QRY access path lets you specify virtuals as key fields. There is no default update processing on this function.

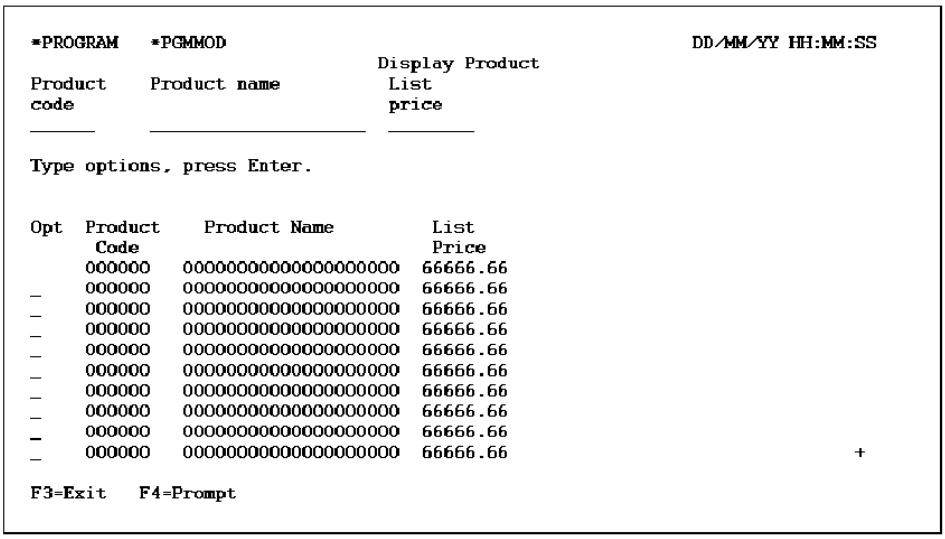
If you modify the action diagram of a DSPFIL function to call a subsidiary function that adds or changes the records on the subfile, the changes do not display unless you force a subfile reload. This can be done by moving the condition *YES to the Subfile Reload field in the PGM context.

You cannot use a DSPFIL function to change or update records unless you add the specific logic to do so.

The DSPTRN function differs from the DSPFIL function in that DSPTRN loads an entire subfile. The DSPFIL function loads only one page of a subfile at a time. For this reason, the SUM, MAX, MIN, and CNT function fields are only valid in the DSPTRN function.

Note: For the same reason, SUM, MAX, MIN, and CNT are also valid in the Edit Transaction (EDTTRN) function and not in the Edit File (EDTFIL) function. To do this type of calculation with a DSPFIL function, you can use a RTVOBJ function to read the records and accumulate the calculation into a work field.

The following is an example of a Display File.



The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	-	Y	R
Part/fully restricted key	I	RST	–	O
Key fields	I	MAP	–	O
Other fields	Any	-/MAP	-	O

Any key field defined to the DSPFIL function as an Input/Mapped parameter acts as a positioner for the subfile display. Any non-key field defined as an Input/Mapped parameter acts as a selector for the subfile.

The following table shows the function options available.

Options	Default Value	Other Values
Subfile selection	Y	N
Subfile end	M(YSFLEND	P,T
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	N	Y
Confirm initial value	M(YCNFVAL)	Y, N
Post confirm pass	N	Y
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N (*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed File I/O Control	M(YDSTFIO)	S, U, N

If the DSPFIL function is attached to an CA 2E access path with a multipart key, the display can be restricted on the major key by passing this key as a restrictor parameter. The effect of this on the display is to move the restricted key field(s) and the associated virtuals onto the subfile control and to hide them on the subfile record.

*Reload Subfile

If you modify the action diagram of the DSPFIL function to call a subsidiary function that adds to or changes the records on the subfile, the changes are not displayed unless you force a subfile reload. This can be achieved by moving the condition *YES to the *Subfile reload field in the PGM context; for example:

```
> USER: command keys

.-CASE                                                          <<<
| -CTL.*CMD key is *Change to 'ADD'                             <<<
| Add new records function                                      <<<
| PGM.*Reload subfile = CND.*YES                                <<<
'-ENDCASE                                                       <<<
```

Post-Confirm Pass Function Option

A Post-Confirm Pass function option is available for this function and can be used to process the subfile records twice. Such a situation might arise if you have added function fields to the screen, which would be validated in the first (pre-confirm) pass. If you then wanted to use these values in further processing, you could specify this in the post-confirm pass.

For more information about:

- Post-confirm pass, see DSPTRN later in this chapter
- On user points, see the chapter "Modifying Action Diagrams"

DSPRCD Device Function

The Display Record (DSPRCD) function defines a program to display a single record from a specified database file. If you do not supply a key value to the function during execution, or if you supply only a partial key, a key value panel prompts you for the key value(s). After supplying the key value(s), the remaining fields in the record appear. All or some of the keys can be supplied as restrictor parameters. If the parameter list contains all the key fields as restrictors, the key panel is bypassed.

All fields on the DSPRCD panel design are output capable only, by default.

You can attach a DSPRCD function to Retrieval (RTV) or Resequence (RSQ) access paths. The access path determines which fields and which records are available for display. There is no default update processing in this function.

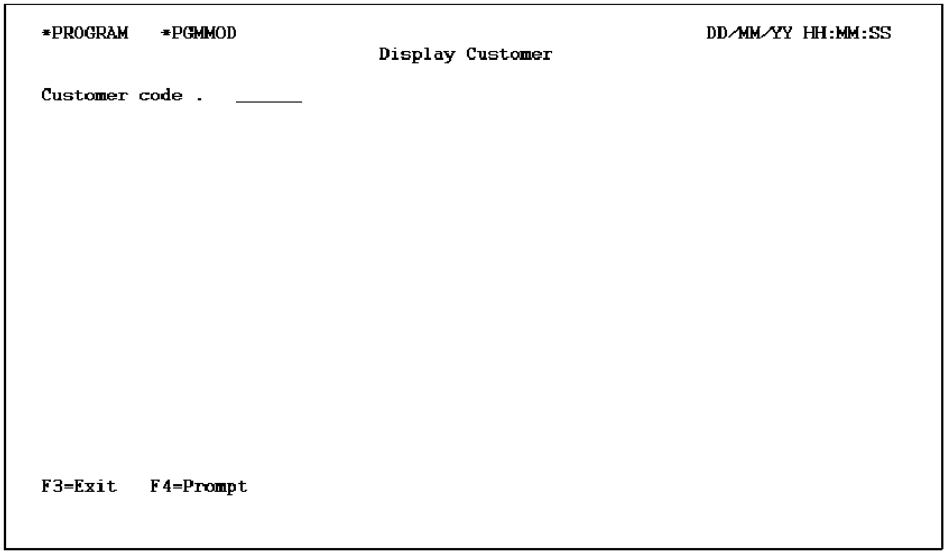
The DSPRCD function executes in *DISPLAY mode only. There are two display panels for this function type: a key panel which prompts for the key values and a detail panel which displays the remaining fields as defined by the device design and the access path.

Design Considerations

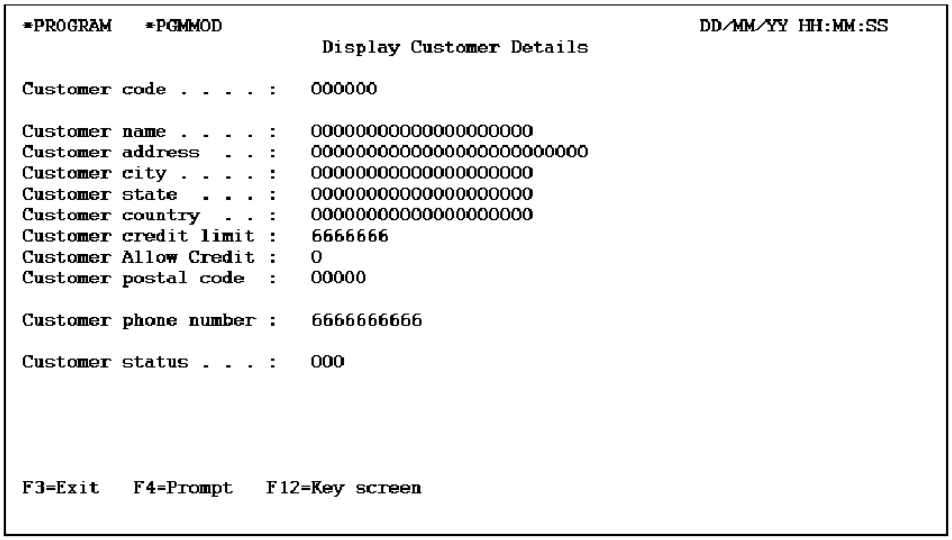
For a single record display panel such as DSPRCD, CA 2E places the fields on the panel design in the following manner.

- Key fields from the based-on access path are placed, one field per line, on both key and detail panels
- Non-key fields from the based-on access path are placed, one field per line, on detail panel designs

The following is an example of a Key panel.



The following is an example of a Detail panel.



The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	-	Y	R
Part/fully restricted key	I	RST	-	O
Other fields	Any	-/MAP	-	O

Parameter fields with a role of Map, which cannot be mapped to any existing field are placed on the display before the other fields.

The following table shows the function options available.

Options	Default Value	Other Values
Confirm prompt	N	Y
Confirm initial value	M(YCNFVAL)	Y, N
Send all error messages	M(YSNDMSG)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information about:

- Function options see the chapter, "Setting Default Options for Your Functions"
- User points see the chapter, "Modifying Action Diagrams"

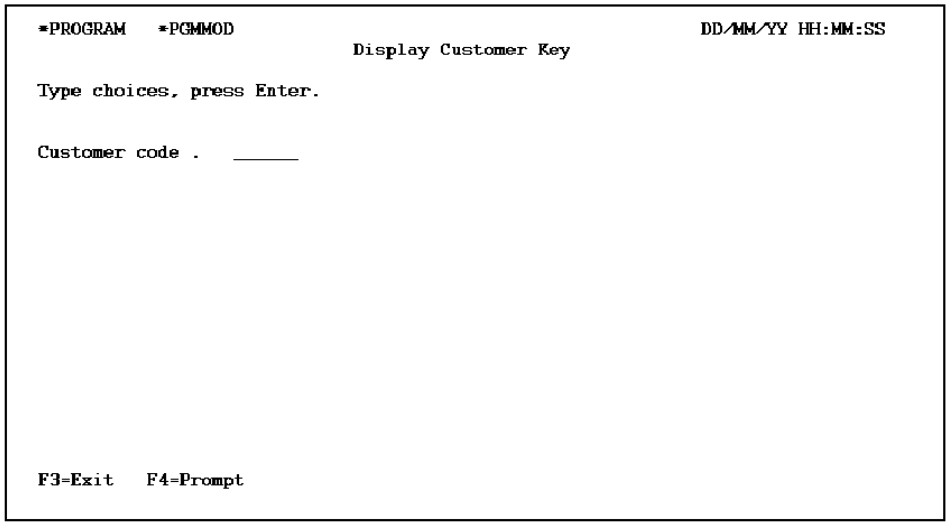
DSPRCD2 Device Function

The Display Record (2 panels) (DSPRCD2) function defines a program that is identical to the Display Record function except that it allows the database record details to extend into two separate display device panels. You can use the scroll keys to move between the panels of details. This function type would be suitable to use with files that contain many fields.

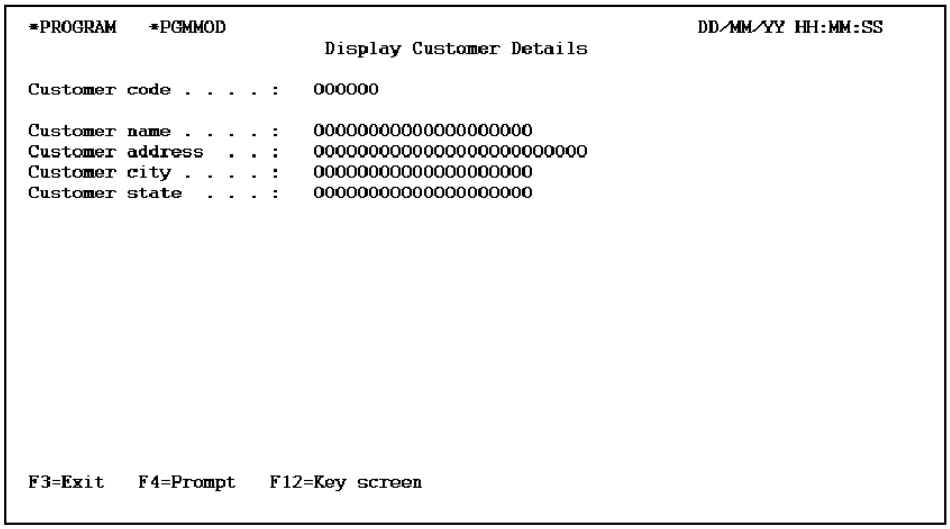
The DSPRCD2 function executes in *DISPLAY mode only. There are three display panels associated with this function type: a key panel that prompts for the key values, and two detail panels which display, by default, all of the fields from the based-on access path. On any panel, you can hide any fields that you do not want to appear. In addition, the same field can appear on more than one detail panel.

This panel is used to display a record that has more fields than currently fit into a single panel. All of the considerations that apply to Display Record also apply to DSPRCD2.

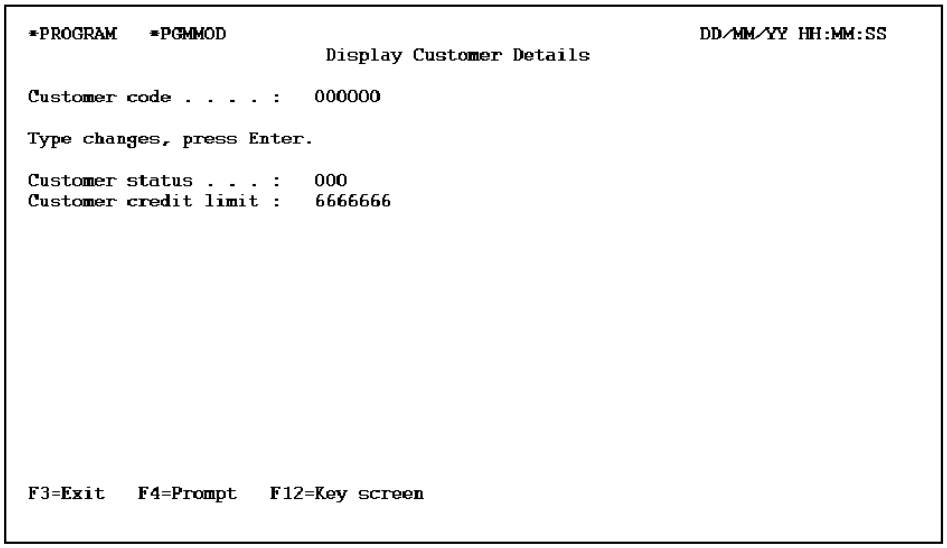
The following is an example of Key panel.



The following is an example of Detail Panel 1.



The following is an example of Detail Panel 2.



Note: For more information, see the Knowledge Base article The working and limitations of EDTRCD2, EDTRCD3, DSPRCD2 and DSPRCD3 function types.

DSPRCD3 Device Function

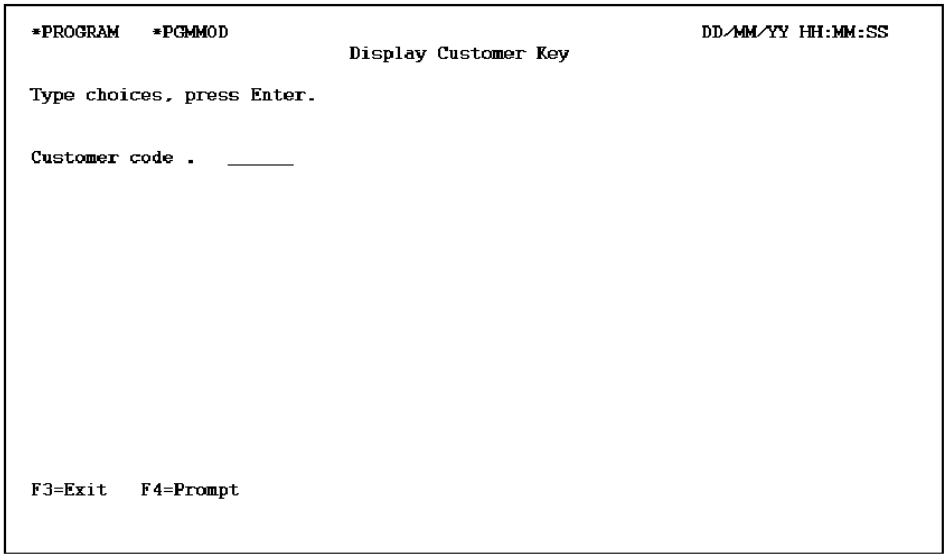
The Display Record (3 panels) (DSPRCD3) function defines a program that is identical to the Display Record function except that it allows the database record details to extend into three separate display device panels. You can use the scroll keys to move between the panels of details. This function type is suitable to use with files that contain many fields.

The DSPRCD3 function executes in *DISPLAY mode only. There are four display panels associated with this function type: a key panel that prompts for the key values, and three detail panels that display, by default, all of the fields from the based-on access path. On any panel, you can hide the fields that you do not want to appear. In addition, the same field can appear on more than one detail panel.

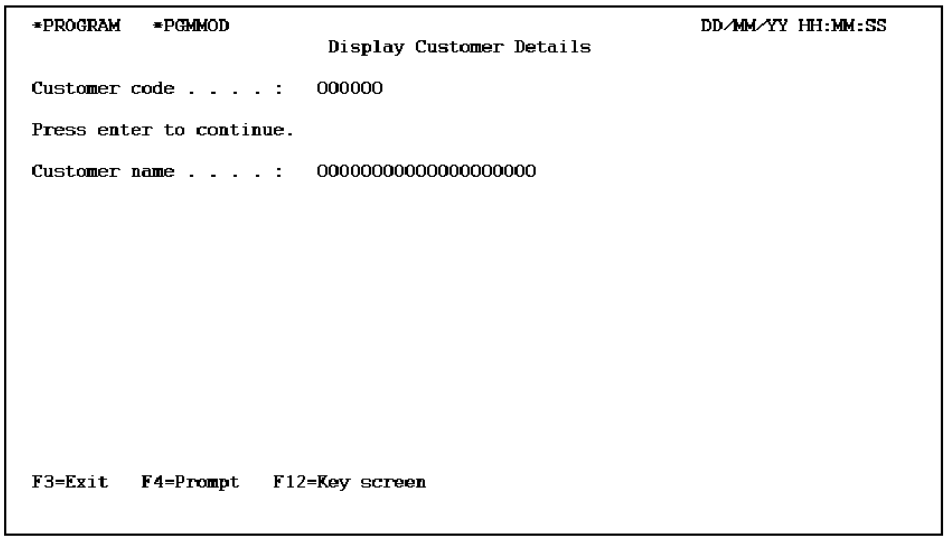
This panel is used to display a record that has more fields than currently fit into a single panel.

All of the considerations that apply to Display Record also apply to DSPRCD3.

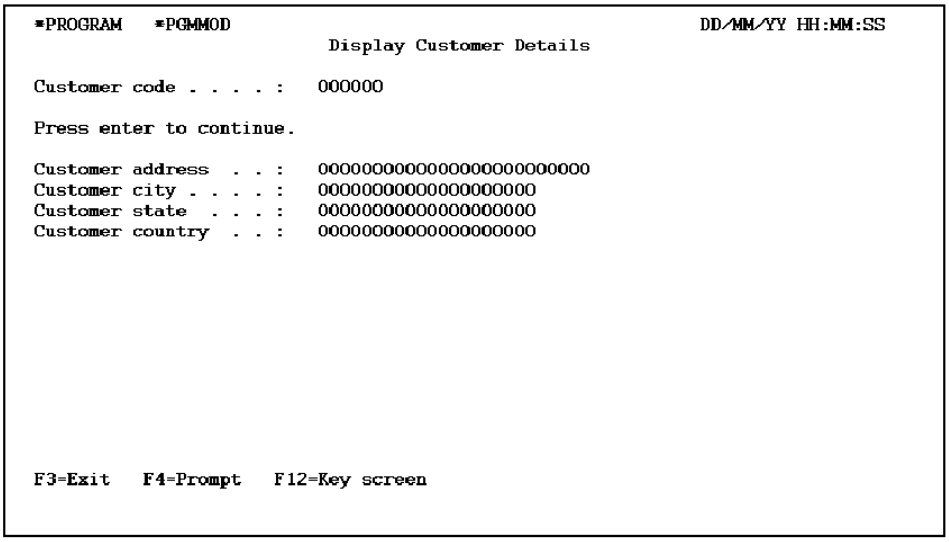
The following is an example of a Key panel.



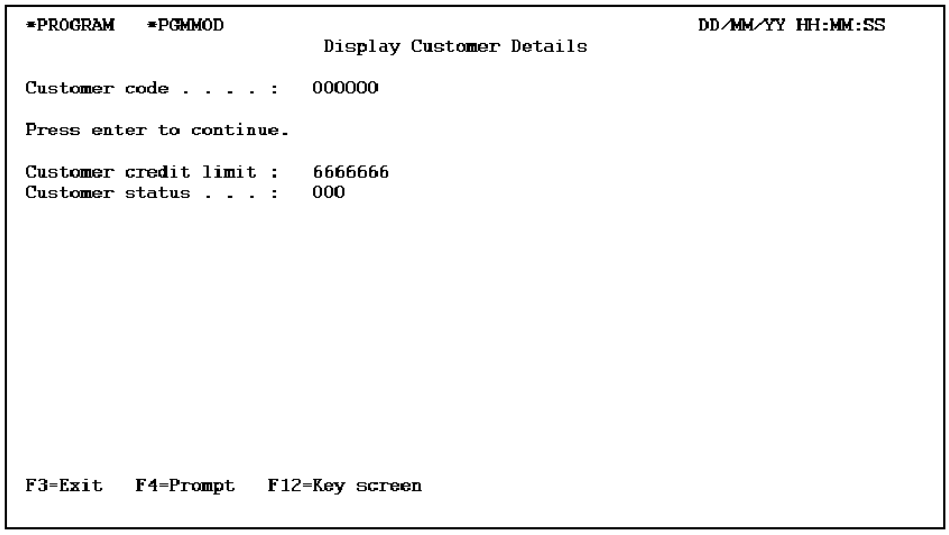
The following is an example of a Detail panel 1.



The following is an example of a Detail panel 2.



The following is an example of a Detail panel 3.



Note: For more information, see the Knowledge Base article The working and limitations of EDTRCD2, EDTRCD3, DSPRCD2 and DSPRCD3 function types.

DSPTRN Device Function

The Display Transaction (DSPTRN) function defines a program that displays the records from two distinct but related database files. The files must connect by a file-to-file relation. The relation that connects the files must be an Owned by or a Refers to relation.

The DSPTRN function has two distinct record formats:

- A header or master record format that corresponds to the owned by or referred to file and is in the subfile control portion of the panel
- A detail record format that corresponds to the owned by, referred to, or referring file and appears as a subfile

The DSPTRN function loads the entire subfile, and is suitable for using SUM, MIN, CNT, and MAX function fields.

A typical use of DSPTRN is to display an Order Header at the top of the panel with a subfile of the associated Order Details below.

The key fields in the header format are input-capable. All non-key fields in the header format are by default output only, .

All fields in the detail format are by default output capable.

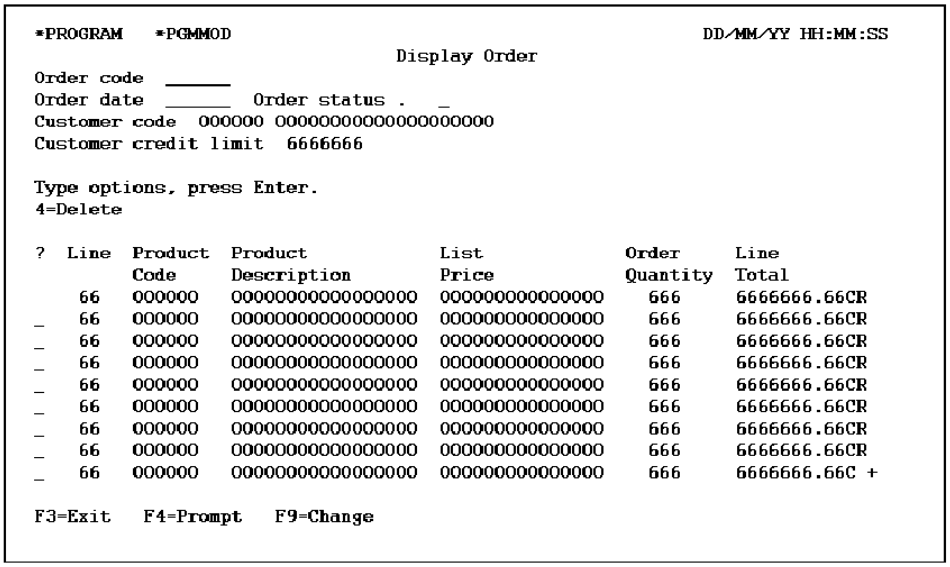
If no key, or a partial key is supplied to the DSPTRN function, the key fields from header format appear, prompting you for the remainder of the key in order to identify the file.

The DSPTRN function must be attached to a Span (SPN) access path. The SPN access path connects two record formats with a common partial key. There is no default update processing for this function type.

In order to be able to create a Span access path:

- An Owned by or Refers to relation must exist between the header and the detail files
- The SPN access path must be created over the owning file or the referred to file
- The access path must be created explicitly to the SPN access path

The following is an example of a DSPTRN panel.



```
*PROGRAM   *PGMMOD                                          DD/MM/YY HH:MM:SS
                                  Display Order
Order code  ______
Order date  ______  Order status .   _
Customer code  000000 00000000000000000000
Customer credit limit  6666666

Type options, press Enter.
4=Delete

?  Line  Product  Product             List              Order     Line
         Code     Description         Price             Quantity  Total
     66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66CR
_    66   000000   00000000000000000   0000000000000000    666     6666666.66C +

F3=Exit   F4=Prompt   F9=Change
```

The panel has the default design for a single and multiple style panel.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	-	Y	R
Part/fully restricted key hdr key	I	RST	-	O
Other fields	Any	-/MAP	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Subfile selection	Y	N
Subfile end	M(YSFLEND)	P, T
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	N	Y
Confirm initial value	M(YCNFVAL)	Y, N
Post-confirm pass	N	Y
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y

Options	Default Value	Other Values
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter "Setting Default Options for Your Functions."

If no key, or a partial key is supplied to the DSPTRN function, the key fields from the header format (which are input-capable) display. These fields prompt for the remainder of the key so that the header record to display can be identified. If the parameter list contains all of the key fields as restrictors, this step is bypassed.

Note: The current implementation of DDS to DDL conversion does not allow RLA functions using Span (SPN) access path and based on the DDS database to work, when the database is converted from DDS to DDL.

Post-Confirm Pass Function Option

A Post-Confirm Pass function option is available for this function, and can be used to process the transaction a second time to carry out additional processing. Such a situation might arise if you have added derived function fields to the control record, calculated in the pre-confirm pass that you want to use in further calculations for each subfile line. You could specify these further calculations in a post-confirm pass.

For example, a function field of type SUM can be added to the header format of a Display Transaction panel to total a value displayed in the detail lines. If a line-by-line percentage of this total is required, it cannot be achieved in one pass. Calculation of the SUM field is only completed at the end of the first (pre-confirm) pass, for example:

```
> USER:  Header update processing
:—
: Total detail lines                                                    <<<
'—
```

```
> USER: Subfile record update processing
:—
: WRK. Pct total = CTL. Total detail lines * RCD. Pct                   <<<
: RCD. Pct total = WRK. Pct total / CON. 100                            <<<
'—
```

For more information on user points, see the chapter "Modifying Action Diagrams."

Automatic Line Numbering

A common requirement when using Edit Transaction functions is to have line numbers for the subfile records issued automatically.

For example, Order and Order Line files could be defined as follows:

```
FIL   Order      CPT   Known by    FLD   Order           CDE
FIL   Order      CPT   Has         FLD   Order date      DTE



FIL   Order line   CPT   Owned by   FIL   Order           CPT
FIL   Order line   CPT   Known by   FLD   Order date      DTE
FIL   Order line   CPT   Refers to  FIL   Product         REF
FIL   Order line   CPT   Has        FLD   Order quantity  QTY
```

If an EDTTRN type function called Edit Order is created over the Order and Order Line files, you might want the order line numbers issued automatically. This can be done as follows:

1. Change the Edit Order function:
    a. Use the Edit Device Design panel to add a function field of type MAX to the order header format (F19), the Highest Line number. This field should be defined as a REF field, based on the line number, so that it calculates the highest line number used so far. Neither the MAX field nor the line number fields need appear on the screen, but can be hidden.
    b. Use the Edit Action Diagram panel to change the call the Create Order Line function so that the Highest Line number field from the CTL context is passed to the Order Line number parameter of the CRTOBJ function.
2. Change the Create Order line function:
    a. Use the Edit Function Parameters panel to change the Order Line number parameter to be a Both parameter rather than Input parameter so that the incremented value is returned to the Highest Line number field.
    b. Use the Edit Action Diagram panel to increment the Highest line number field by one before writing to the database. Return the incremented value to the order line number parameter after writing the record to the database.

For more information on user points, see the chapter "Modifying Action Diagrams."

EDTFIL Device Function

The Edit File (EDTFIL) function defines a program to maintain records in a file, many at a time, using a subfile. The subfile is loaded a page at a time when you press the scroll keys.

For each key field on the subfile record, there is an equivalent field on the subfile control header format. Key fields can be used as positioner parameters in the subfile control format to specify which fields are displayed in the subfile. The subfile has, by default, all records in the access path as input-capable.

Effects of Parameters

- If the high order keys are restrictor fields (RST), only records with keys matching the parameters display.
- If the high order keys are mapped positioner fields (POS), subfile records display on the first subfile page starting with the key values that you entered. If the subfile control field is output only or hidden, it functions as a restrictor field.
- Parameter fields that have a role of MAP are placed on the subfile control format and not on the subfile record format.

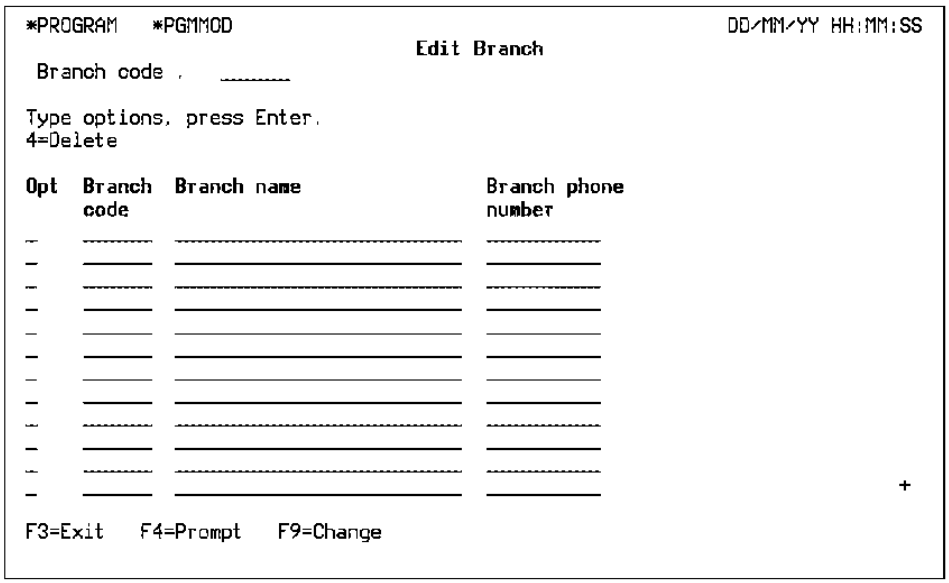
An EDTFIL function can be attached to a Retrieval (RTV) or a Resequence (RSQ) access path.

If specified in the function options, an EDTFIL function contains calls to the Create Object, Delete Object, and Change Object functions. To remove functions associated with these actions, change the function options. You can also modify the default functions called from the Edit File by editing the action diagram.

For more information on function options, see the chapter, "Modifying Function Options."

If the EDTFIL function is in *ADD mode, an empty subfile appears. If it is in *CHANGE mode, a page of records from the based-on access path appear as modifiable fields. You can change the default function logic for this function type by modifying the action diagram, and changing the function options. In the resultant program, you can alternate between *ADD and *CHANGE mode by pressing F9.

The following is an example of an EDTFIL panel.



The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	-	Y	R
Part/fully restricted key	I	RST	–	O
Other fields	Any	-/MAP	_	O

The following table shows the function options available.

Options	Default Value	Other Values
Create record	Y	N
Change record	Y	N
Delete record	Y	N
Dynamic program mode	Y	N
Subfile selection	Y	N
Subfile end	M(YSFLEND)	P, T
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	Y	N
Confirm initial value	M(YCNFVAL)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M (YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information about:

- Function options see the chapter "Setting Default Options for Your Functions"
- User points see the chapter "Modifying Action Diagrams"

EDTRCD Device Function

The Edit Record (EDTRCD) function defines a program that maintains records in a specified file, one record at a time.

The EDTRCD function executes in either *ADD or *CHANGE mode. There are two display panels for this function type: a key panel that prompts for the key values and a detail panel that displays all the fields from the based-on access path.

The EDTRCD function has the following default logic:

- It accepts the key panel (or key values if the key panel was bypassed) and checks for record existence
- In *ADD mode, it displays a panel with blank input-capable fields if the record does not exist
- In *CHANGE mode, it retrieves the record through a RTV or RSQ access path and displays the record as modifiable, input-capable fields

If no key value, or a partial key value, is supplied as a restrictor parameter, the key panel prompts for the remainder of the key. By specifying the elements of a composite key as restrictor (RST) parameters, the key panel is bypassed and the function exits when the record is changed and Enter is pressed.

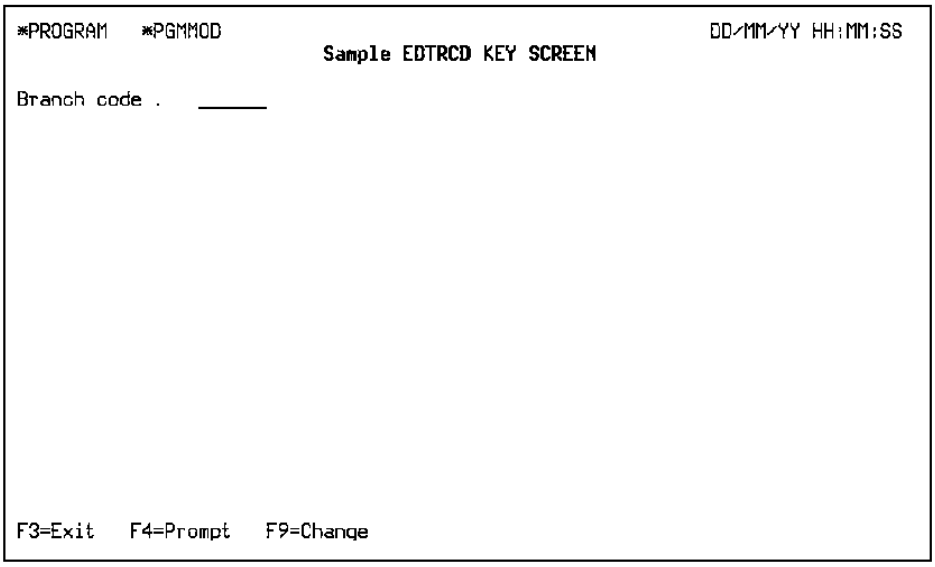
The EDTCD function is in *CHANGE mode when first called unless there are no records existing in the file. If you set the function option Dynamic program mode to Y, the function automatically chooses the correct mode. You can toggle between *ADD and *CHANGE modes by pressing F9 on the key panel.

An EDTRCD function can be attached to a Retrieval (RTV) or a Resequence (RSQ) access path.

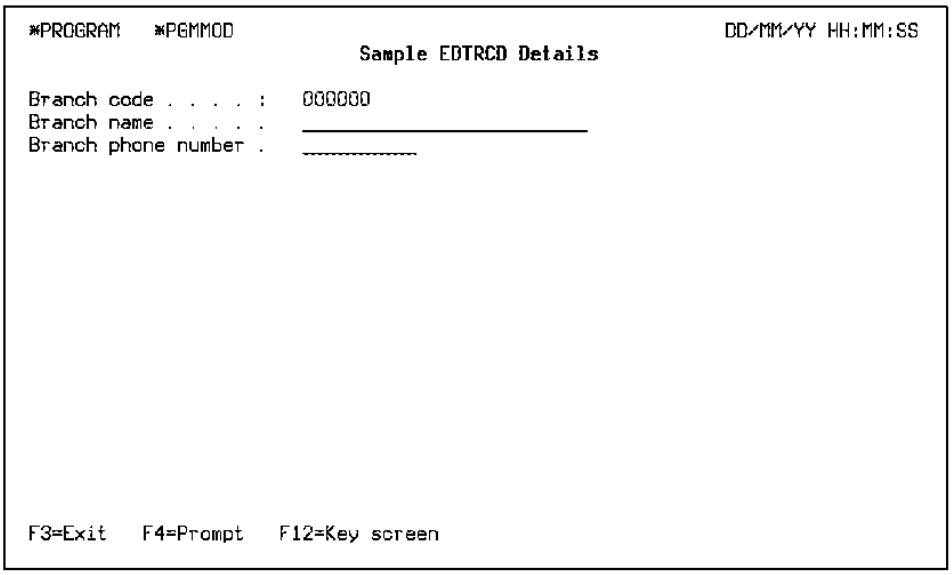
By default, an EDTRCD function contains calls to the CRTOBJ, DLTOBJ, and CHGOBJ functions based on the function options.

You can disable these calls by changing the function options.

The following is an example of a Key panel.



The following is an example of a Detail panel.



The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	-	Y	R
Part/fully restricted key	I	RST	-	O
Other fields	Any	-/MAP	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Create record	Y	N
Bypass key screen	N	Y
Exit after add	N	Y
Change record	Y	N
Delete record	Y	N
Dynamic program mode	N	Y
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	Y	N
Confirm initial value	M(YCNFVAL)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information about:

- Function options, see the chapter "Setting Default Options for Your Functions"
- User point, the chapter, "Modifying Action Diagrams"

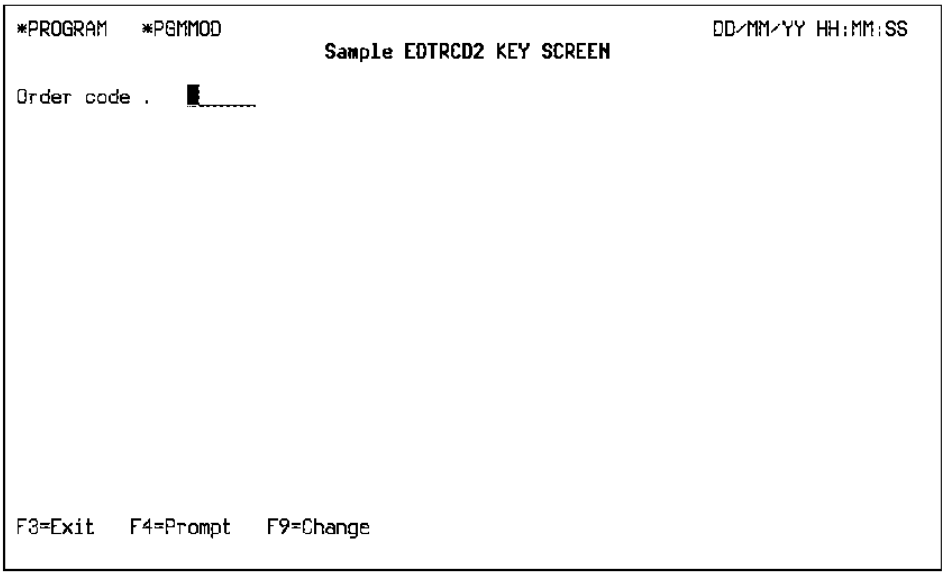
EDTRCD2 Device Function

The Edit Record (2 panels) (EDTRCD2) function is identical to the Edit Record function except that it allows the record details to extend to two separate display panels. You can use the scroll keys to move between the pages of detail. This function type is suitable for files containing many fields.

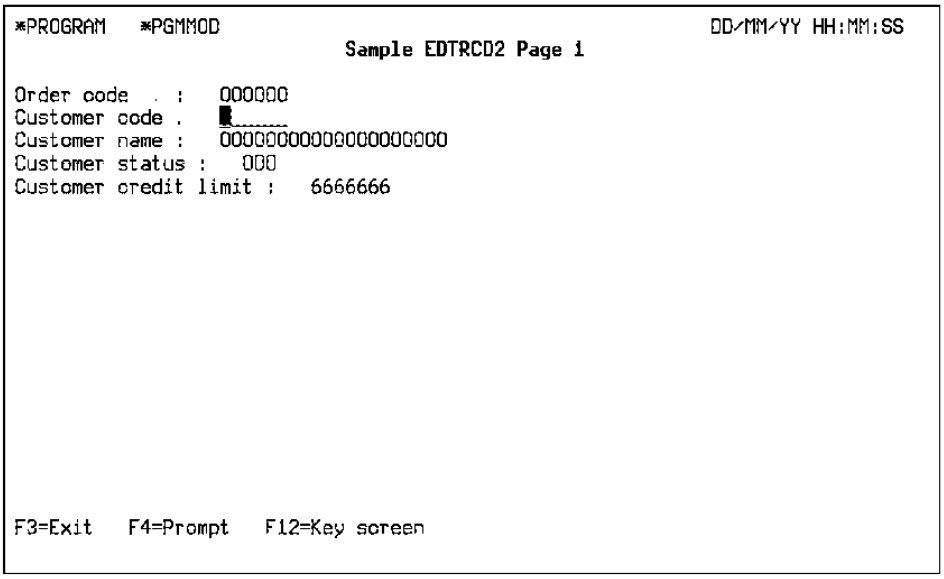
The EDTRCD2 function executes in either *ADD or *CHANGE mode. There are three display panels for this function type: a key panel that prompts for the key values, and two detail panels that display, by default, all of the fields from the based-on access path. On any panel, you can hide fields that you do not want to appear. In addition, the same field can appear on more than one detail panel.

All of the considerations that apply to the Edit Record function also apply to EDTRCD2.

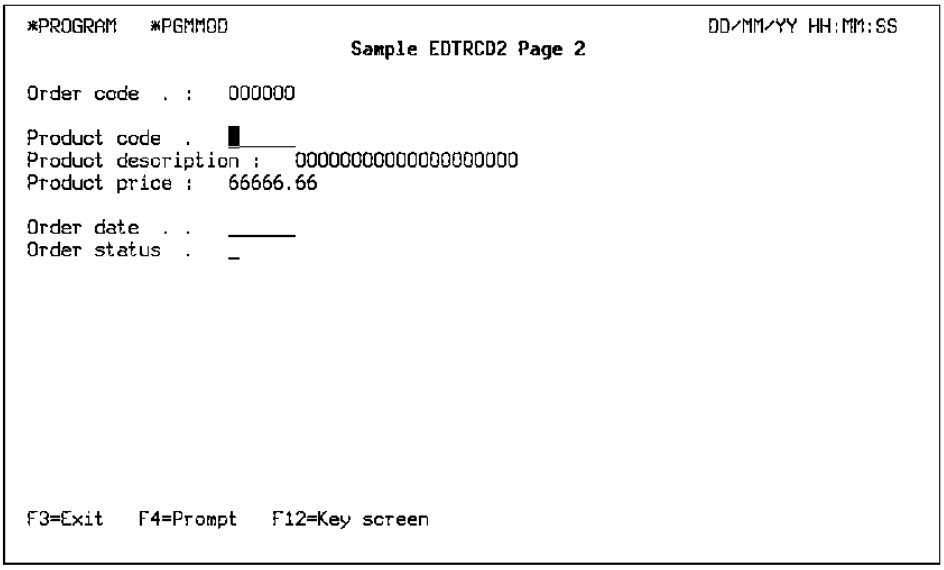
The following is an example of a Key panel.



The following is an example of a Detail panel 1.



The following is an example of a Detail panel 2.



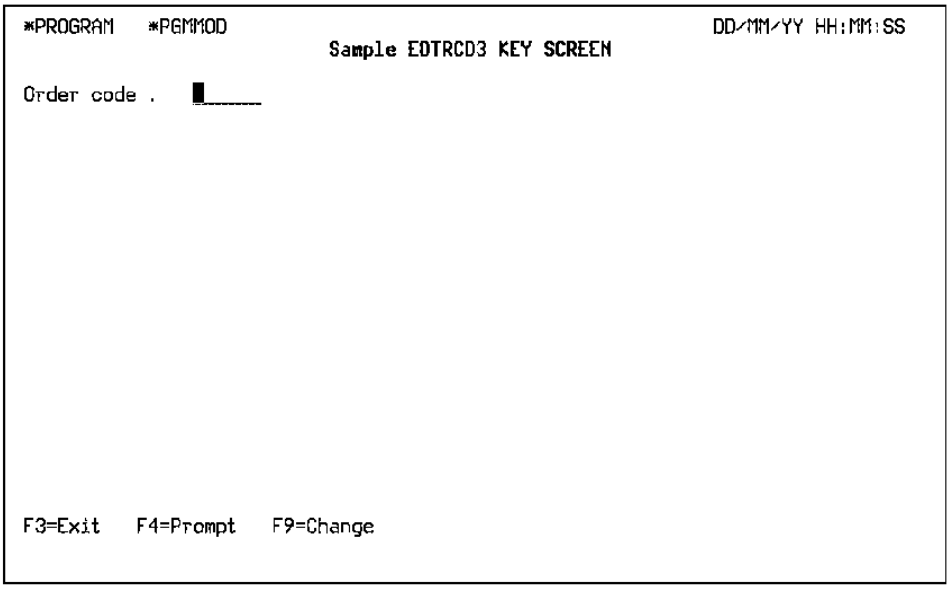
Note: For more information, see the Knowledge Base article The working and limitations of EDTRCD2, EDTRCD3, DSPRCD2 and DSPRCD3 function types.

EDTRCD3 Device Function

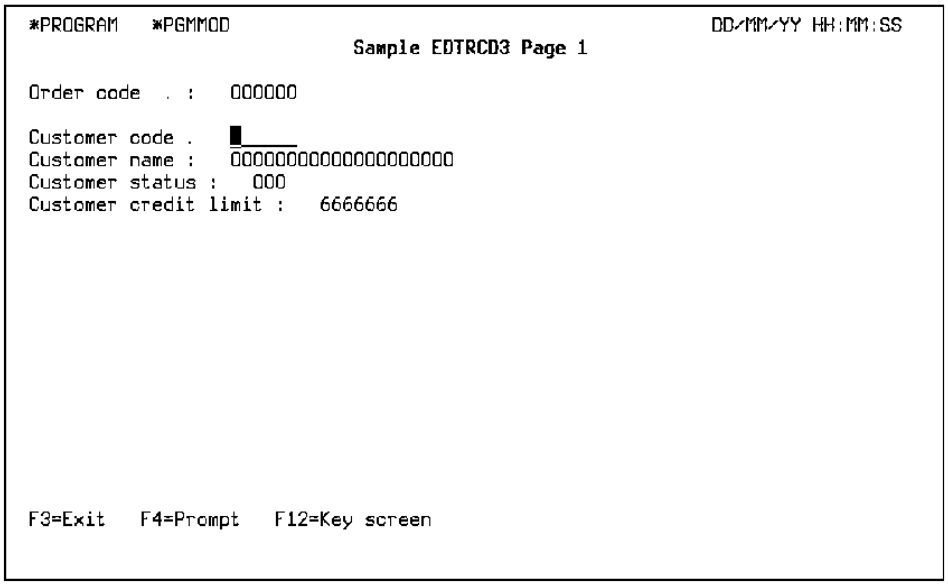
The Edit Record (3 panels) (EDTRCD3) function is identical to the Edit Record function except that it allows the record details to extend to three separate display panels. You can use the scroll keys to move between the panels of detail. This function type is suitable for files containing many fields.

The EDTRCD3 function executes in either *ADD or *CHANGE mode. There are four display panels for this function type: a key panel that prompts for the key values and three detail panels that display, by default, all of the fields from the based-on access path. On any panel, you can hide fields that you do not want to appear. In addition, the same field can appear on more than one detail panel.

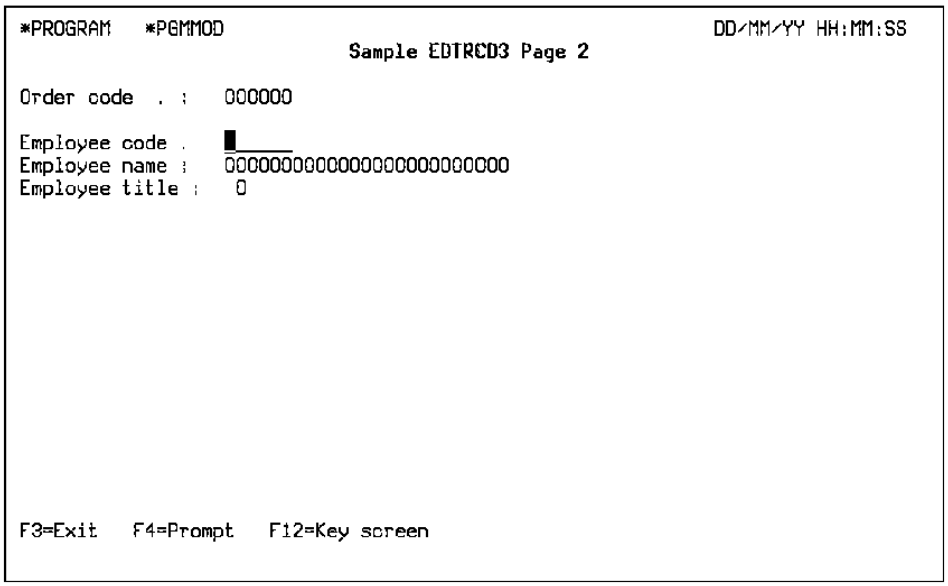
All of the considerations that apply to the Edit Record function also apply to EDTRCD3.



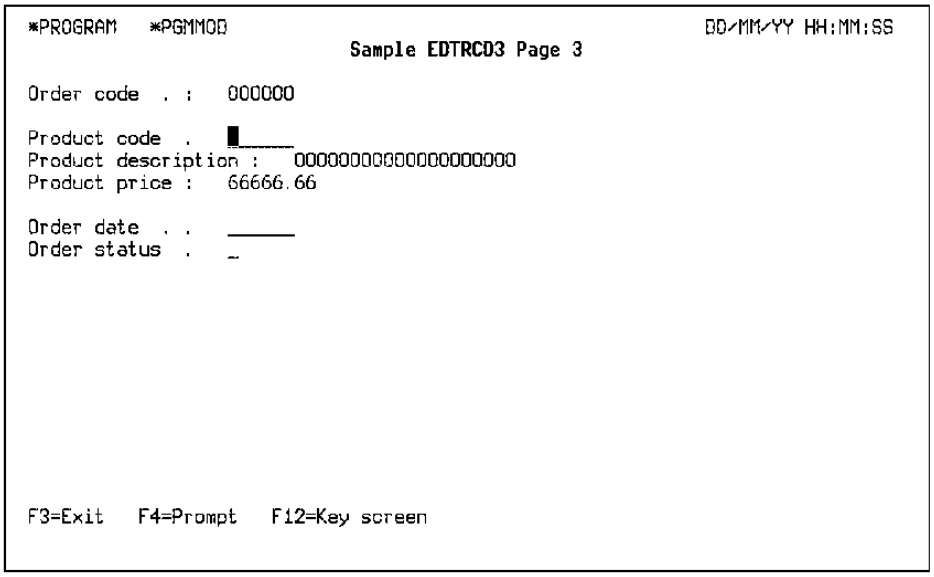
The following is an example of a Detail panel 1.



The following is an example of a Detail panel 2.



The following is an example of a Detail panel 3.



Note: For more information, see the Knowledge Base article The working and limitations of EDTRCD2, EDTRCD3, DSPRCD2 and DSPRCD3 function types.

EDTTRN Device Function

The Edit Transaction (EDTTRN) function defines a program that maintains the records on a specified pair of header and detail files. The files must be connected by a file-to-file relation. The relation that connects the files must be an Owned by or a Refers to relation.

The EDTTRN function has two distinct record formats: a header, or master record format that corresponds to the owning or referred to file and is in the subfile control portion of the panel; and a detail record format that corresponds to the owned by, or referring file, and appears as a subfile. The EDTTRN function loads the entire subfile, and is suitable for using SUM, MIN, CNT, and MAX function fields.

The EDTTRN function has the following default function logic:

- In *ADD mode, the header keys are accepted as parameters, if provided. By specifying the keys as restrictor (RST) parameters, the key fields are output only on the panel. The panel appears with blank input-capable fields in the header master file and detail subfile record formats. You can then enter data in the header record format and the detail record format subfile for validation.
- In *CHANGE mode, the header keys are accepted as parameters. If no key, or a partial key is supplied to the EDTTRN function, the key fields from the header format display to prompt for the remainder of the key. If the parameter list contains a fully restricted key, this step does not occur.
- All detail records matching the header record keys are loaded into the subfile. The panel displays the header record file and the first page of the detail file as a subfile. You can add, change, or delete subfile records as you want. After successful validation the file is updated.

An EDTTRN function includes calls to the CRTOBJ, DLTOBJ, and CHGOBJ functions by default for both header and detail formats. These default functions are included in the action diagram for the function. To remove this default processing, you change the function options. These functions use the associated Update (UPD) access path update. There can be six separate calls to these internal functions: three calls for the header format file; three calls for the detail format file.

The EDTTRN function must be attached to a Span (SPN) access path. The SPN access path connects two record formats with a common partial key.

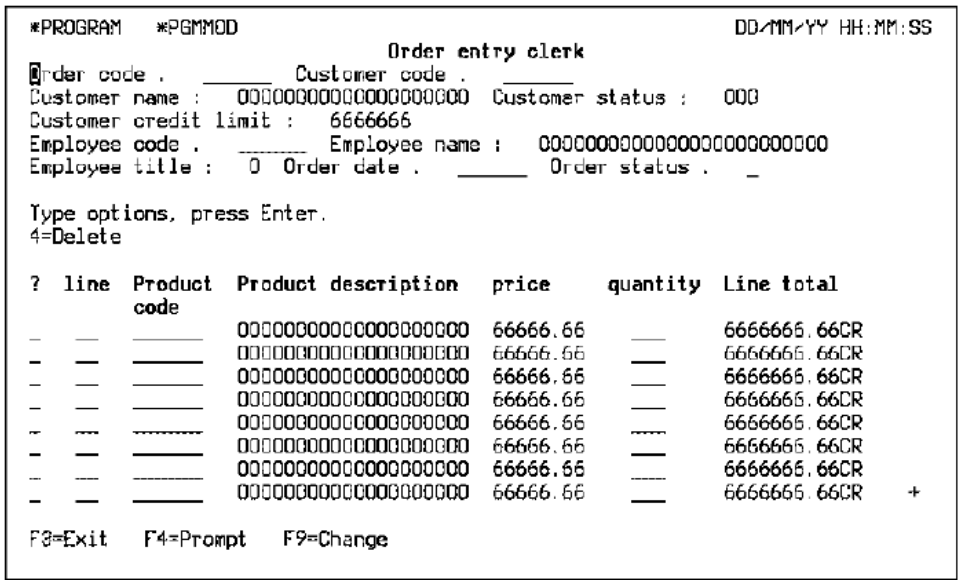
In order to be able to create a Span access path:

- An Owned by or Refers to relation must exist between the header and the detail files
- The SPN access path must be created over the owning file or the referred to file
- The access path formats must be added explicitly to the SPN access paths

The typical use of an EDTTRN is to display an Order Header at the top of the panel with a subfile of the associated Order Detail below.

The EDTTRN function differs from the EDTFIL function in that the EDTTRN function loads an entire subfile. The EDTFIL function only loads one page of a subfile at a time.

The following is an example of an Edit Transaction panel.



The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return Code	B	-	Y	R
Part/fully restricted key hdr key	I	RST	-	O
Other fields	Any	-/MAP	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Create transaction	Y	N
Change transaction	Y	N
Create record	Y	N
Delete record	Y	N
Delete transaction	Y	N
Dynamic program mode	N	Y
Subfile selection	Y	Y, N
Subfile end	M(YSFLEND)	P, T
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	Y	N
Confirm initial value	M(YCNFVAL)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	N	Y
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U

Options	Default Value	Other Values
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O control	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter "Setting Default Options for Your Functions."

Automatic Line Numbering

A common requirement when using Edit Transaction functions is to have line numbers for the subfile records issued automatically.

For example, Order and Order Line files could be defined as follows:

```
FIL    Order           CPT    Known by        FLD    Order           CDE
FIL    Order           CPT    Has             FLD    Order date      DTE
FIL    Order line      CPT    Owned by FIL    Order          CPT
FIL    Order line      CPT    Known by FLD    Order date     DTE
FIL    Order line      CPT    Refers to       FIL    Product         REF
FIL    Order line      CPT    Has             FLD    Order quantity  QTY
```

If an EDTTRN type function called Edit Order is created over the Order and Order Line Files, you might want the order line numbers issued automatically. This can be done as follows:

1. Change the Edit Order function:
    a. Use the Edit Device Design panel to add a function field of type MAX to the order header format (F19), the Highest Line number. This field should be defis a REF field, based on the line number, so that it calculates the highest line number used so far. Neither the MAX field or the line numbers fields appear on the screen, but can be hidden.
    b. Use the Edit Action Diagram panel to change the call the Create Order Line function so that the Highest Line number field from the CTL context is passed to the Order Line number parameter of the CRTOBJ function.
2. Change the Create Order line function:
    a. Use the Edit Function Parameters panel to change the Order line number parameter to be a Both parameter rather than Input parameter so that the incremented value is returned to the Highest Line number field.
    b. Use the Edit Action Diagram panel to increment the Highest Line number field by one before writing the database. Return the incremented value to the order line number parameter after writing the record to the database.

For more information on user points, see the chapter "Modifying Action Diagrams."

Note: The current implementation of DDS to DDL conversion does not allow RLA functions using Span (SPN) access path and based on the DDS database to work, when the database is converted from DDS to DDL.

EXCEXTFUN User Function

The Execute External Function (EXCEXTFUN) allows you to specify a high-level program using an action diagram. You can also use an EXCEXTFUN as a user-defined *Notepad function that can serve as a repository of standardized action diagram constructs that you can easily copy into the action diagrams of other functions.

For more information on user-defined *Notepad functions, see Using NOTEPAD in the chapter "Modifying Action Diagrams"

The function is implemented as a separate program and has its own action diagram. The function can be created with an access path of *NONE. You can submit an EXCEXTFUN function for batch processing from within an action diagram.

The EXCEXTFUN has no default logic. It presents an empty action diagram. Any function or set of functions of any type can be included in an EXCEXTFUN and compiled so that it can be executed as a program.

For documentation purposes, the EXCEXTFUN can be optionally attached to Retrieval (RTV), Resequence (RSQ), or Update (UPD) access paths.

A good practice is to have only a single action diagram construct with an EXCEXTFUN. This single construct is a call to an internal function (EXCINTFUN) that contains all the functionality required. Both functions define the same parameters. This way, you can choose to reference an EXCEXTFUN (function you can call) or EXCINTFUN without duplicating the action diagram constructs.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	–	Y	R
Other fields	Any	–	–	O

Parameters of role Varying (VRY) are allowed for this function type.

The following table shows the function options available.

Options	Default Value	Other Values
Send all error messages	M(YSNDMSG)	Y, N
Confirm prompt	Y	N
Confirm initial value	M(YCNFVAL)	Y, N
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N

Options	Default Value	Other Values
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Overrides if submitted job	*	F
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter, "Setting Default Options for Your Functions."

Using Batch Processing

The following information is an example of how to use an EXCEXTFUN to process records using a batch process:

Assume the following:

- Each Order has a status containing a condition that reflects the current processing stage of the order
- Orders for which payments have yet to be received have an Order Status of Unpaid
- Each Order also has an associated Customer Code identifying the customer who placed the Order, and an Order Total Value indicating the amount due for the Order
- The requirement to process all unpaid Orders and to update the corresponding customers unpaid value (held in the Customer file) with the total amount owed by the customer
- Each Order, once processed, has its status changed to Processed

Use the RTVOBJ function, Process All Unpaid Orders, to process the records in the Order file. The access path attached to the RTVOBJ determines the order in which the records on the order file are read.

Assume that the Retrieval access path of the Order file is keyed by Order Number. It would not be advisable to use this access path for the RTVOBJ as there is no need to process all the Orders in the Order file.

Consequently, you can create and use alternative access paths such as the following:

1. A Retrieval (RTV) access path with select or omit criteria selecting only Unpaid Orders.
2. A Resequence (RSQ) access path keyed by Order status and Customer code.

The second access path is preferable because a restrictor parameter of Order Status with a supplied condition EQ Unpaid for the RTVOBJ has the equivalent select or omit effect on the records retrieved.

Furthermore, having the Customer code as the secondary key presents Unpaid Order records for each Customer in sequence. This allows the change of Customer code to indicate the end of a list of Order records belonging to a customer.

This allows the total unpaid value for a Customer to be accumulated and the corresponding Customer record updated only when all Unpaid Orders for that Customer are processed resulting in a significant reduction in database I/O.

The logic required in the RTVOBJ, Process All Unpaid Orders, is as follows:

```
 .> USER: Process Data record
 ..—
...-CASE
.. |-DB1.Customer code EQ WRK. Saved Customer code
.. | Continue to accumulate ...
..| WRK.Cust. value unpaid Orders = WRK.Cust. value unpaid orders
.. |                               + DB1. Order total value
.. |-*OTHERWISE
.. | Update Cust.unpaid value — Customer *
.. | | Customer code             = WRK.Saved Customer code
.. | B Cust. value unpaid Orders = WRK.Cust. value unpaid Orders
.. | Save new customer code ...
.. | WRK.Saved Customer code = DB1.Customer code
.. | Reset accumulator ...
.. | WRK.Cust. value unpaid Orders = DB1.Order total value
..  '-ENDCASE
.. Set Order status to 'P' (Still unpaid but Customer updated) ...
.. Change Order — Order *
..         | Order number             = DB1.Order number
..         | Customer code            = DB1.Customer code
..       | Order status            = CND.Unpaid(Customer updated)
..         | Order total value       = DB1.Order total value
.'—
```

Since the value for a customer is only updated for a change in Customer, the last customer record must be accounted for as follows:

```
. USER: Exit processing
..—
.. Update last Customer ...
.. Update Cust. unpaid value - Customer *
..    I  Customer code              = WRK.Saved Customer code
..    B  Cust. value unpaid orders = WRK.Cust. value unpaid order
.'—
```

The CHGOBJ Change Order is the default CHGOBJ for the function with no additional user logic.

The CHGOBJ Update Cust. unpaid value has the following user logic to ensure that the total value is accumulated onto what is already present for the customer on file as follows:

```
> USER: Processing after Data read
 .—
 . Add to Customers unpaid value ...
 . PAR. Cust. value unpaid Orders = PAR.Cust. value unpaid Orders
 .                   + DB1.Cust. value unpaid Orders
 '—
```

Once the functions and logic have been defined, the RTVOBJ, Process All Unpaid Orders, can be embedded within an EXCEXTFUN Batch process, Unpaid Orders as follows:

```
> Batch process unpaid orders
 .—
 .  Restrict to process only UNPAID orders (i.e. Order status = 'U') ...
 .  Process all unpaid orders - Orders *
 .     I   RST Order status= CND.Unpaid
 '—
```

After generating and compiling the EXCEXTFUN, it can be executed in batch using the i OS SBMJOB or equivalent command. You can choose to execute this command from another function, such as a PMTRCD acting as a menu, by using the Execute Message function to submit the function to batch.

For more information about:

- Submitting jobs from within an action diagram, see Submitting Jobs Within an Action Diagram in the chapter "Modifying Action Diagrams"
- Using the Execute Message function, see EXCMSG Message Function later in this chapter

EXCINTFUN User Function

The Execute Internal Function (EXCINTFUN) allows you to specify a portion of an action diagram that can be used repeatedly in other functions. You can also use an EXCINTFUN as a user-defined *Notepad function, which can serve as a repository of standardized action diagram constructs that you can easily copy into the action diagrams of other functions.

For more information on user-defined *Notepad functions, see Using NOTEPAD in the chapter "Modifying Action Diagrams."

The EXCINTFUN is implemented as inline source code (or a macro function) within the source code of the calling function. That is, whenever you make a call to this function type, it is embedded in the source code of the calling function at the point where you made the call.

You cannot attach an EXCINTFUN to an access path. When defining this function type specify *NONE for the access path.

You might use the EXCINTFUN to perform a calculation routine that will be used repeatedly within a function or several functions.

Try to encapsulate as much of each action diagram as possible in the EXCINTFUN rather than in native code. This encourages functional normalization and improves maintainability. In addition, it improves the time it takes to load the action diagram of the referencing functions.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Any field	Any	-	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Execution location	W	S
Generate as subroutine?	N	Y
Share subroutine	M(YSHRSBR)	N, Y

Example

An example of EXCINTFUN could be in performing a calculation, such as working out a percentage that you want to call several times in different functions. You could:

1. Define an EXCINTFUN, attached to a database file, with access path *NONE.
2. Define input parameters of the function, such as Number and Total and the output parameters, such as Percentages.
3. Define actions required in the supplied action diagram.

EXCMSG Message Function

The CA 2E Execute Message (EXCMSG) function allows a request message to be executed by the calling function. A request message is generally a CL (control language) command.

You enter the request string in the second-level text of the message function. The command can be executed in the iSeries native environment.

The EXCMSG function is attached to a special CA 2E shipped system file called *Messages.

To implement the EXCMSG function, you define an EXC type message function. Once you define the command to execute, you insert a call to the EXCMSG function from a user point within the action diagram of a calling function. The EXCMSG function is then implemented as a call to a CL program supplied by CA 2E.

The following table shows the parameters available.

Parameter	Usage	Role	Default	Option
Return code	B	-	Y	R
Message id	I	-	Y	R
Message data	I	-	Y	R
Field to receive message text	I	-	-	0

The following table shows other properties.

Options	Default Value	Other Values
Message severity	20	00-99
Second level text	-	System request

The EXCMSG function is implemented by a call to a CL program. The default environment is controlled by the model value YEXCENV, but can be overridden for individual Execute Message functions. The text of the message can be tailored to the environment.

It is not possible to use the i OS override commands: Override Database File (OVRDBF), Override Display file (OVRDSPF), and Override Print File (OVRPRTF) with the EXCMSG function because the resulting overrides are only in effect in the invocation level of the implementing CA 2E CL program.

If you use the i OS commands OPNDBF and OPNQRYF as the request message text, you must specify a value of TYPE(*PERM) for these commands in order to prevent the closure of the file on return from the implementing CA 2E CL program.

You can invoke the i OS command prompt by pressing F4 after entering the command string. Although CA 2E allows you to use message substitution data variables, (&1, &2), within the string, i OS does not accept these values within the string. To overcome this restriction replace the ampersand (&) symbol with an at (@) symbol.

Advantage of SBMJOB over Execute Message

Alternatively, you can submit EXCEXTFUN, EXCUSRPGM, and PRTFIL functions for batch execution from within an action diagram. This method has the following advantages:

- Numeric parameters can be passed
- The complexities of constructing the submit job command string are hidden
- References to submitted functions are visible by CA 2E impact analysis facilities.

For more information on submitting jobs from within an action diagram, see Building Applications, Submitting Jobs Within an Action Diagram in the chapter, "Modifying Action Diagrams."

Specifying EXCMSG

To specify an EXCMSG function, define an EXC type message function:

1. At the Edit Message Function Details panel, type **Z** and press F7 to access second level text.

   The Edit Second Level Message Text panel appears.

2. Specify the name of the command to execute. If you are unsure of parameters, you can prompt for the names by pressing F4.

EXCUSRPGM User Function

The Execute User Program (EXCUSRPGM) function allows you to specify the connection to a user-written high-level language program that is called by CA 2E functions. You can specify parameters on the call to this function. You can also submit an EXCUSRPGM function for batch processing from within an action diagram.

You can attach the EXCUSRPGM function to any access path or you can specify *NONE for the access path. You should normally attach the EXCUSRPGM function to a file containing some or all of the function parameters.

You cannot specify Neither (N) type parameter for this function type; however, you may specify Varying (VRY) role parameters. There is no *Return Code parameter for this function type unless you explicitly define it.

There is no action diagram or device design associated with this function type.

When you create an EXCUSRPGM function, CA 2E assigns a source member name for the program. You can override this to be the name of the HLL program that you want to call. You need to do this if the program already exists and you are now defining it to the model.

You should declare all the parameters required by your EXCUSRPGM function. Parameters can be declared in the normal fashion using the Edit Function Parameters panel. However, you should first determine the domain of the parameter fields in your user-written program to ensure that they correspond to the parameters of your calling function.

Example

The EXCUSRPGM function can be used to call IBM supplied programs such as the QDCXLATE program to translate a string.

The program translates a given string of a given length to uppercase using the QSYSTRNTBL table. A system-supplied table carries out lowercase to uppercase conversion.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Any field required by program	I/O/B	-/VRY	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Execution location	W	S

Options	Default Value	Other Values
Generate error routine	M(YERRRTN)	Y, N
Overrides if submitted job	*	F

EXCUSRSRC User Function

The Execute User Source (EXCUSRSRC) function specifies either:

- User-written high-level language source code is to be included within the source code generated by a calling function.
- Device language statements, for example, DDS that can be applied to a device function to customize the associated device design.

For more information on device user source, see this module, in the chapter, "Modifying Device Designs," Device User Source topic.

The HLL source code of the EXCUSRSRC function must be the same as that of the calling function; that is, a function implemented in RPG can only call an EXCUSRSRC function that is RPG.

You can attach the EXCUSRSRC function to any access path, or you can specify *NONE for the access path. You should normally attach the EXCUSRSRC function to a file containing some or all of the function parameters. If there are no suitable files, it may be worth considering defining a dummy file using a Defined as relation.

For example:

```
FIL    User source    REF    Defined as    FIL    User source    REF
```

Overall User Source Considerations

You cannot specify Neither (N) type parameters for this function type. The following table shows the allowed parameters.

Parameters	Usage	Role	Default	Option
Any field	I/O/B	-	-	O

No action diagram, device design, or function options are associated with this function type.

Although the EXCUSRSRC function provides flexibility, for ease of maintenance, you should use action diagram programming whenever possible.

The HLL source is held in the source member that CA 2E assigns. CA 2E provides a source member name which you can override with the name of the source member that you want to call. Change the source name at the Edit Function Details panel.

The source member must reside in the appropriate source file. The source file must be in the library list of any job that generates source for CA 2E functions that call the EXCUSRSRC function. The files are as follows:

- For RPG functions, the source member must reside in the file QRPGSRC.
- For COBOL functions, the source member must reside in the file QCBLSRC or QLBLSRC.

Parameters must be passed in accordance with the instance code that you generate in the EXCUSRSRC function. To define parameters for the user source:

- Define the EXCUSRSRC function and optionally attach it to an existing access path or else specify *NONE for the access path.
- Type P next to the function from the Edit Functions panel and define the parameters accordingly.

For more information on defining parameters, refer to this module, in the chapter, "Modifying Function Parameters."

Substitution Variables

The following substitution variables let you embed source generation information into user source that is then resolved into the actual values in the final program source. One use of this feature is to define generic preprocessing programs.

Variable Name	Description
*MBR	Program source member name
*FILE	Program source file name
*LIB	Program source file library name
*TYPE	Source type (*RPG/*COBOL)

RPG Source Considerations

You can include all RPG specification types in the source member, except an H specification. CA 2E sorts the different specification types into their appropriate positions within a program:

- CA 2E codes the source in the normal order for RPG specifications, which is:
  - H specification (obtained from model values YRPGHSP for RPG, YRP4HSP for RPGIV Program, and YRP4HS2 for RPGIV Module)
  - F and K specifications
  - E specifications
  - I specifications
  - C specifications
  - O specifications
- CA 2E treats the first RPG calculation specifications, which are not part of a subroutine, as the instance code. If there are repeated calls to an EXCUSRSRC function, CA 2E generates the code on every call to the function at the point indicated by the action diagram.

CA 2E automatically inserts any C-specifications that follow a subroutine, but are not part of a subroutine, in the ZZINIT subroutine.

The order of specification is

- Instance code C-specifications
- C-specifications for subroutines called by instance code
- Initialization C-specifications

Parameter fields are only recognized in the factor one, factor two, and result positions of the RPG calculation specifications that form part of the instance code.

Parameters must take the form #UMMMM where U is the parameter usage defined on the function (I, O, or B), and MMMM is the mnemonic name (DDS name) of the formal parameter. For example:

```
C           #IORVL    ADD       #BLNVL      #OTLVL
```

CA 2E checks usage of parameters within the instance code both for correspondence with the formal parameter and for use within the user RPG code. For example if field ORCD is alphanumeric, CA 2E would not allow the following:

```
C                   Z-ADD         ZERO        #BORCD
```

The letter U is reserved as the initial letter for user-specified field names and subroutine names in user-written source. If you use other prefixes, you may find they conflict with CA 2E generated names in certain instances.

If your EXCUSRSRC function contains an ICOPY statement, ensure that the source member to be copied does not contain a subroutine. CA 2E generates the EXCUSRSRC code as a subroutine in your function. The ICOPY includes the other subroutine inside the EXCUSRSRC subroutine. This causes compile errors as a subroutine cannot be embedded in another subroutine.

If you use RPG indicators, you must ensure that they do not conflict with those used elsewhere in the program. To ensure that your usage of indicators in user-written source code does not conflict with the use made of them in CA 2E generated code, you should save the current indicator values at the start of user-written code and restore them at the end. For example, to save and restore indicators 21 to 40:

```
C         MOVEA*IN,21     UWIN 20           * Save
C
C         .....user code.....
C
C         MOVEAUWIN       *IN,21            * Restore
```

COBOL Source Considerations

You can include all COBOL statement types in the source member, except LINKAGE SECTION code and additional PROCEDURE DIVISION USING statements. CA 2E places the different divisions and sections in the user source in their appropriate positions within a program.

You must begin the code for each division or section according to the standard COBOL conventions. For example:

```
+ + + + - A 1       B.. ... 2 ... ... 3
                 WORKING-STORAGE SECTION.
                 01  VAR-1   PIC 9999 COMP.
                 01  VAR-2   PIC 9999 COMP.

                 PROCEDURE DIVISION.
                 MOVE VAR-1 TO VAR-2
                 PERFORM    ROUTINE
                 MOVE VAR-2 TO VAR-1
```

In the generated code, Relevant Area B code normally follows that code generated by CA 2E from the user source function but do not assume it does. You will find that FILE-SECTION entries, in particular, precede entries generated for other function types. Code is thus considered to be divided by anything found in Area A.

CA 2E treats Area B code between a PROCEDURE DIVISION Area A statement (or the beginning of the member) and the next Area A statement (or the end of the member if sooner) as instance code. If there are repeated calls to an EXCUSRSRC function, CA 2E generates this code on every call to the function, at the point indicated by the action diagram. CA 2E includes all other code in the generated program only once, if it is a section.

CA 2E does not actually incorporate the following Area A statements from user source into the generated code because they are all present in the generated code already. If they are specified in EXCUSRSRC code, they serve only to show where the subsequent Area B statements (that is, all preceding next Area A statements or the end of the member if sooner) should be placed:

```

```

```
+ + + + + - A 1    B.. ...2 ... ...3
                IDENTIFICATION DIVISION.
                ENVIRONMENT DIVISION.
                CONFIGURATION SECTION.
                INPUT-OUTPUT SECTION.
                FILE-CONTROL.
                I-O CONTROL.
                DATA DIVISION.
                FILE-SECTION.
                WORKING-STORAGE-SECTION.
```

CA 2E decodes and places Area A statements as follows:

- SPECIAL-NAMES: Area A statements are decoded and the constituent SPECIAL NAMES, if any, are placed in the correct place in the generated code. The actual words SPECIAL NAMES are omitted, as they are already present in the generated code.
- COMMITMENT CONTROL FOR: Area A statements are decoded and the constituent file names placed in the correct place in the generated code. The actual words COMMITMENT CONTROL FOR are placed in the generated code, unless they are already present because of use of commitment control by another function.
- CA 2E assumes that a user source section headed USR-INIT. consists of code to be placed in the ZZINIT (initialization) SECTION. The USR-INIT. heading is omitted from the generated code.
- Sections in the user source that follow a DECLARATIVES Area A statement are placed, with their Area A section headings, in the correct place in the generated code. The actual words DECLARATIVES and END DECLARATIVES are placed in the generated code, unless they are already there because of another function using this COBOL facility.
- All other code sections, with their Area A headings, are placed in the generated code following all non-user source generated sections.

If an EXCUSRSRC function contains code for more than one section, remember that these sections are not consecutive within the host program; they are only required to be syntactically correct within their own section or division. If you use the COBOL syntax checker when editing user source, you may incur errors because the syntax checker assumes that the code present in the source member constitutes a complete COBOL program, and that the sections and divisions encountered are consecutive within the source. Both these assumptions are false for the source of an EXCUSRSRC function.

The COBOL user source must be correct within the context of the source into which it is inserted. If you are generating COBOL 85, it must follow COBOL 85 conventions; if you are generating COBOL 74 it must follow COBOL 74 conventions. Keep these points in mind:

- For COBOL 85, CA 2E generated code contains explicit scope terminators such as END-IF and END-PERFORM. Within inline code, you must use scope terminators rather than full stops. In code that is part of another division or section and contextually independent, either full stops or scope terminators can be used.

- Because explicit scope terminators are not available in COBOL 74, you cannot use them in EXCUSRSRC functions.

You can include an EXCUSRSRC function written in COBOL 74 within a function generated in COBOL 85, provided the COBOL 74 code does not include any language elements that are invalid in COBOL 85. First copy the COBOL 74 source to the COBOL 85 source file (QLBLSRC) and do one of the following:

- Convert the source according to the previous conventions. This means replacing appropriate full stops with scope terminators.
- Enclose the EXCUSRSRC function call within an action diagram sequence construct (IS - insert sequence) so that it is implemented as a contextually independent subroutine.

You can place parameter fields at any point in the instance code. You cannot split them across a source record boundary. Parameters must be of the form USR-PARM-U-MMMM where U is the parameter usage defined in the function (I, O or B) and MMMM is the mnemonic name (DDS name) of the formal parameter. For example:

```
+ + + + - A 1      B .. ... 2 ... ... 3
                ADD USR-PARM-I-ORVL TO USR-PARM-B-
                LNVL
                GIVING USR-PARM-O-TLVL
                END-ADD
```

CA 2E checks the usage of parameters within the instance code, the U part of the name, for correspondence with the formal parameter. The letter U is reserved as the initial letter for user-specified field and section names in user-written source. If you use other prefixes, you may find they conflict with CA 2E generated names in certain instances. If you use COBOL indicators, you must ensure that they do not conflict with those used elsewhere in the program.

EXCUSRSRC Function Example

Say you want to define a user-written HLL function called Get customer credit limit into another CA 2E standard function. The EXCUSRSRC function has three parameters:

IOB	Parameter	GEN Name
I	Customer code	CUCD
I	Trial value	TRQT
O	Trial limit	TRLM

To specify this function you might do the following:

1. Define the three-parameter fields as fields using the Define Objects panel.
2. Define an EXCUSRSRC function using the Edit Functions panel. The function can be attached to any file with an access path value of *NONE.
3. Specify the three parameters for the EXCUSRSRC function using the Edit Function Parameters panel.
4. From the Edit Function Details panel for the EXCUSRSRC function, change the program name for the function to the name of the source member containing the user-written code.
5. Code the EXCUSRSRC function in the nominated source member.

   Refer to the following examples of RPG and COBOL and then proceed to Step 6.

   The following example shows RPG EXCUSRSRC.

```
** Get customer credit limit                      ) <- IGNORED
** Company      : Universal Sprocket Co           )
** System       : Widget processing system        )
** Author       : YOU                             )
FUUCUCRLOIF       E    K    DISK                  ) <- OTHER SPECIFICATIONS
 * Customer credit limits                         )

 * Set up parameters                              ) <- INSTANCE CODE
C                   MOVE #ICUCD  UACUNB           )
C                   Z-ADD#ITRQT  UATRQT           )
C                   EXSR UACRLM                   )
C                   Z-ADDUACRLM  #OTRLM           )
CSR UACRLM          BEGSR                         ) <- SUBROUTINE
*================================================ )============================
* Get credit limit                               )
*================================================ )============================
C                   MOVEA*IN,60  UWIN  1          ) *Save
* Set of record error indicators.                )
C      UACUNB       CHAIN @ CUCRQQ             60 )
C      *IN60        IFEQ '0'                      )
C      UATRQT       ADD QQCRLM   UACRLM           )
C      END                                        )
C         _        MOVEA UWIN    *IN,60           ) *Restore
*================================================ )============================
CSR    UAEXIT       ENDSR                         )

* Initialization for credit test                 )
*================================================ )============================
C                   Z-ADD*ZERO   UACRLM           ) <- ZZINIT code
*================================================ )============================
```

   The following example shows COBOL EXCUSRSRC.

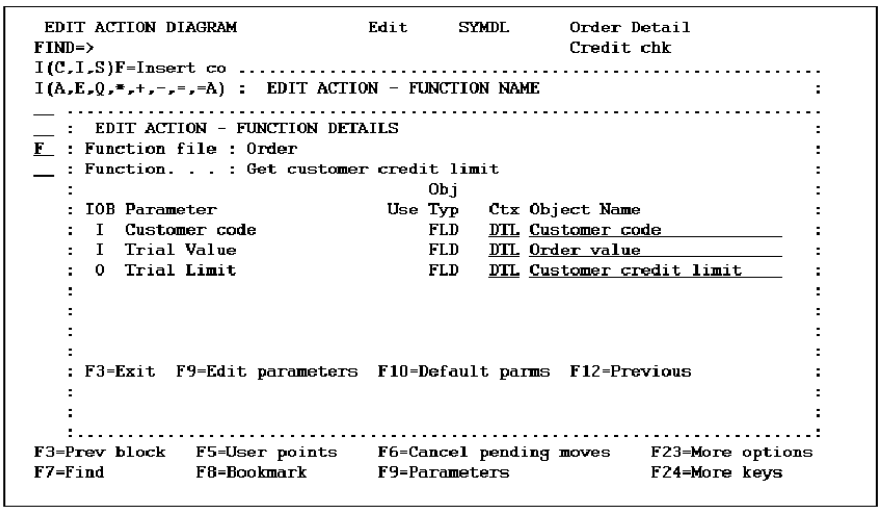
```
```

```
+++++ -A 1 B .. ... 2 ... ... 3
  ** Get customer credit limit                              ) <- IGNORED
  ** Company      : Universal Sprocket Co                   )
  ** System       : Widget processing system                )
  ** Author       : FRED                                    )
     INPUT-OUTPUT SECTION.                                  ) <- OTHER
     FILE-CONTROL.                                          )     SPECIFICATIONS
       SELECT UUCUCRL0                                      )
       ASSIGN TO DATABASE-CBABREL1                          )
       ORGANIZATION IS INDEXED                              )
       ACCESS MODE IS DYNAMIC                               )
       RECORD KEY IS EXTERNALLY-DESCRIBED-KEY               )
       FILE STATUS is FILE-STATUS.                          )
  * Customer credit limits                                  )
  FILE-SECTION.                                             )
  FD UUCUCRL0                                               )
     LABEL RECORDS ARE STANDARD.                            )
  01 UUCUCRL0-F.
   COPY DDS-ALL-FORMATS OF UUCUCRL0.

  WORKING-STORAGE SECTION.                                  ) <- NEW VARIABLES
  01 UWIN  PIC X.                                           )

  PROCEDURE-DIVISION.                                       ) <- INSTANCE CODE
  * Set up parameters                                       )
   MOVE USR-PARM-I-CUCD TO UACUNB OF YCUCRQQ                )
   MOVE USR-PARM-I-TRQT TO UATRQT OF YCUCRQQ                )
   PERFORM UACRLM                                           )
   MOVE UACRLM OF YCUCRQQ TO USR-PARM-O-TRLM                )

  * Get credit limit                                        ) <- SUBROUTINE
  UACRLM SECTION.                                           )
   MOVE IND(60) TO UWIN                                     )
  * Set of record error indicators.                         )
   READ UUCUCRL0                                            )
    FORMAT IS 'YCUCRQQ'                                     )
   END-READ                                                 )
   IF C-NO-RECORD                                           )
      MOVE C-IND-ON TO IND(60)                              )
   ELSE                                                     )
      MOVE C-IND-OFF TO IND(60)                             )
   END-IF                                                   )

   IF IND(60) = C-IND-ON                                    )
      ADD UATRQT OF YCUCRQQ, QQCRLM                         )
       GIVING UACRLM OF YCUCRQQ                             )
      END-ADD                                               )
   END-IF                                                   )
```

1. In the action diagram of the function from which you want to call the Get customer credit limit function, insert an action:

```
> USER: User defined action
: -
:   Get customer credit limit *                          < < <
'-
```

Supply the required parameters for the action using the Edit Action Details panel. For example:



The following table shows an example of the code generated for the call. These are the names of the values passed to the parameters.

IOB	Parameter	GEN Name	Variable	GEN Name
I	Customer code	CUCD	Customer code	CUCD
I	Trial value	TRQT	Order value	ORQT
O	Trial limit	TRLM	Customer credit limit	CULM

CA 2E generates the following RPG and COBOL code for the particular call:

```
UAEXIT.                                              )
 EXIT.                                               )

* ZZINIT code.
USR-INIT SECTION.                                    ) < - ZZINIT CODE
 MOVE ZEROES TO UACRLM OF YCUCRQQ                    )

The following example shows EXCUSRSRC - RPG call.



* Get customer credit details
* Set up parameters
C               MOVE # 1CUCD        UACUNB
C               Z-ADD #2ORQT        UATRQT
C               EXSR UACRLM
C               Z-ADDUACRLM         #1CULM

The following example shows EXCUSRSRC - COBOL Call.



+++ -A 1 B.. ... 2 ... ... 3
   * Get customer credit details
   * Set up parameters
          MOVE Z1CUCD OF ZSFLRCD-WS-O TO UACUNB OF YCUCRQQ
          MOVE Z2ORQT OF ZSFLCTL-WS-O TO UATRQT OF YCUCRQQ
          PERFORM UACRLM
          MOVE UACULM OF YCUCRQQ TO Z1CULM OF ZSFLRCD-WS-O
```

CA 2E includes the other source code statements for the EXCUSRSRC function into the source of the calling function without modification.

MAX Function Field

MAX (maximum) is a special type of field usage that is assigned to function fields containing the result of a computation of the highest value for a particular field.

The MAX field usage uses a special built-in routine that computes the value of the function field based on the input parameter. For instance, within an Edit Transactions function type, you could define a MAX function field on the header record file to calculate the highest value of orders that appear in the detail file. You must define the input and result parameters associated with the MAX function field. The MAX function field would calculate the resultant value on initialization and call to the function.

MAX function fields have two, and only two, parameters:

- **A result parameter**—This is the MAX function field itself. You must place this field on the totaling format of any function (Display Transactions or Edit Transactions) that calls the MAX function field.
- **An input parameter**—This specifies the field for which MAX determines the highest value. This field must be present on the detail record format of the calling function.

Function fields of usage MAX must always be numeric. If the function field is defined as a referenced (REF) field based on another numeric field, CA 2E assumes that the based-on field is the field whose maximum value is calculated.

Examples

The following are usage examples of the MAX function field:

- Largest order item: maximum of order quantity
- Highest line number: maximum of line number

Function Field

MINMIN (minimum) is a special type of field usage assigned to the function fields containing the result of a computation of the lowest value for a particular field.

The MIN field usage uses a special built-in routine that computes the value of the function field based on the input parameter. For instance, within an Edit Transactions function type, you define a MIN function field on the header record file to calculate the lowest value of orders appearing in the detail file. You must define the input and result parameters associated with the MIN function field. The MIN function field calculates the resultant value on initialization and call to the function.

MIN function fields have two, and only two, parameters:

- **A result parameter**—This is the summed MIN function field itself. You must place this field on the totaling format of any CA 2E function (Display Transactions or Edit Transactions) that calls the MIN function field.
- **An input parameter**—This specifies the field for which MIN determines the lowest value. This field must be present on the detail record format of the calling function.

Function fields of usage MIN must always be numeric. If the function field is defined as a referenced (REF) field based on another numeric field, CA 2E assumes that the based-on field is the field whose minimum value is calculated.

Example

The following are usage examples of the MIN function field:

- Smallest order item: minimum of order quantity
- Lowest line number

MTRCD Device Function

PThe Prompt Record (PMTRCD) function defines a program to prompt for a list of fields defined by a specified access path. You can pass the validated fields as parameters to any other function whether it is interactive, batch, a user-defined function to print a report, or an i OS message function such as Execute Message.

There is no default action for this function type. If you want to call another function once you validate the PMTRCD fields, you must specify this in the action diagram.

The fields that appear, by default, on PMTRCD are based on the file/access path over which the function is built. You can drop all panel format relations and fields and use your own function fields and subsequent validation on the function as an alternative to validating fields with file relations.

You can attach a PMTRCD function to a Retrieval (RTV) or a Resequence (RSQ) access path. There is no update processing for this function unless you specifically include it in the action diagram. A Prompt Record function does not validate the existence of a particular record from the underlying access path. It will, however, validate panel level relations and date and time fields.

The PMTRCD function executes in *ENTER mode only and there is only one display panel for this function.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	–	Y	R
Fields from access path	Any	-/MAP	-	O
Any other fields	Any	-/MAP	-	O

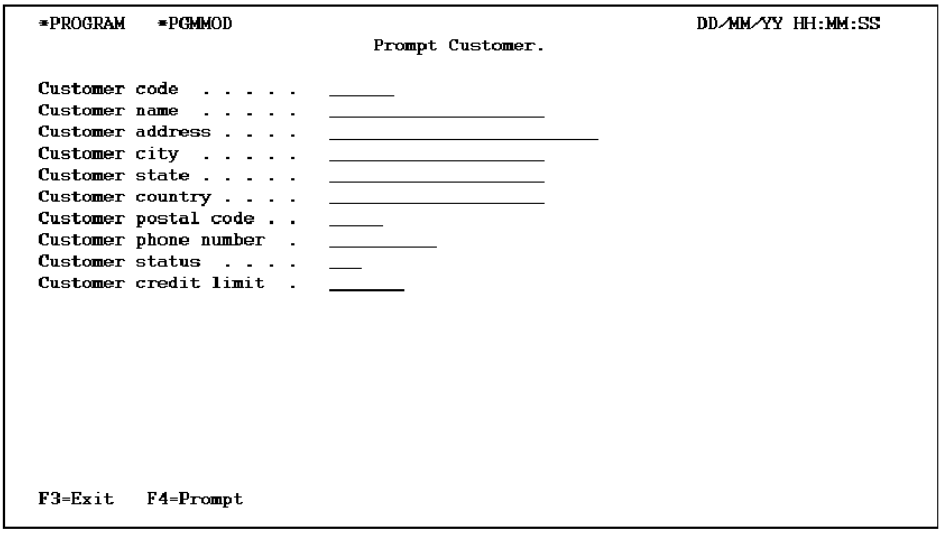
The following table shows the function options.

Options	Default Value	Other Values
Repeat prompt	N	Y
Confirm prompt	Y	N
Confirm initial value	M(YCNFVAL)	Y, N
Send all error messages	M(YSNDMSG)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)

Options	Default Value	Other Values
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, 0
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distribute a file I/O Control	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter, "Setting Default Options for Your Functions."

The following is an example of Prompt and Record panel.



For more information on user points, see the chapter, "Modifying Action Diagrams."

PRTFIL Device Function

The Print File (PRTFIL) function defines a program to print the records from a specified access path. The PRTFIL function prints all records in a single file hierarchy of the based-on access path and provides header and total formats for key fields. You can submit a PRTFIL function for batch processing from with an action diagram.

Most considerations that affect PRTFIL also affect Print Object (PRTOBJ).

Default Processing

The following describes default processing:

- You can specify up to 24 levels of totaling.
- You can print records from more than one access path by embedding Print Object functions within this function type. You can view the overall structure of the report in the Edit Device Structure panel.
- You can attach a PRTFIL function to a Retrieval (RTV), a Resequence (RSQ), or a Query (QRY) access path. The Query access path allows you to specify virtuals as key fields. You can omit certain records from printing by setting the *Record Selected Program field to No.

Device Considerations

The following are device considerations:

- The default design for a PRTFIL function includes standard header, top of page, first page, detail final totals, and footer formats.
- If there is more than one key field, CA 2E creates a header and total format for each additional key level. By using the Edit Report Formats panel, you can drop unwanted formats except detail standard header/footer formats.
- The formats belonging to any embedded PRTOBJ functions are present on the PRTFIL device design; however, you cannot alter them. You can only specify the indentation level of the embedded PRTOBJ formats within the report structure.

Parameter Considerations

Effect of restrictor parameters:

- If you furnish all the keys of the access path to which a PRTFIL function attaches as restrictor parameters to the function, only the record with the given key or keys prints.
- If you furnish only some of the keys (such as the major keys) as restrictor parameters, all of the records with the given key print.
- If you furnish none of the keys of the access path as restrictor parameters, all of the records on the access path print.

Effect of positioner parameters:

- If you furnish all of the keys of the access path to which a PRTFIL function attaches as positioner parameters to the function, only the records with a key value greater than or equal to the given key or keys print.
- If you furnish only some of the keys (such as the major keys) as restrictor parameters, but some or all of the remaining keys are passed as positioner values, only those records with keys equal to the restrictor values and greater than or equal to the positioner values print.
- If none of the keys of the access path are supplied as positioner parameters, all of the records in the access path within the specified restrictor group print.

Level breaks:

- CA 2E defines a level break whenever a major key value changes.
- On a level break, the Print File resets the fields in the associated controlling Header format and its associated Total format. The fields are reset to blank, zero, or values from the DB1 context as appropriate. The total number of formats for the PRTOBJs and PRTFILs cannot exceed 104 if generated in RPG and 96 if generated in COBOL.
- You can remove a level break and the associated processing by dropping the associated header and total formats. Totals automatically accumulate at the higher levels.
- If you hide the formats, the processing remains in the execution but the output is suppressed from the report.
- You can add a maximum of 24 PRTOBJ functions to one PRTFIL function.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	–	Y	R
Key fields	I	RST/POS	-	O

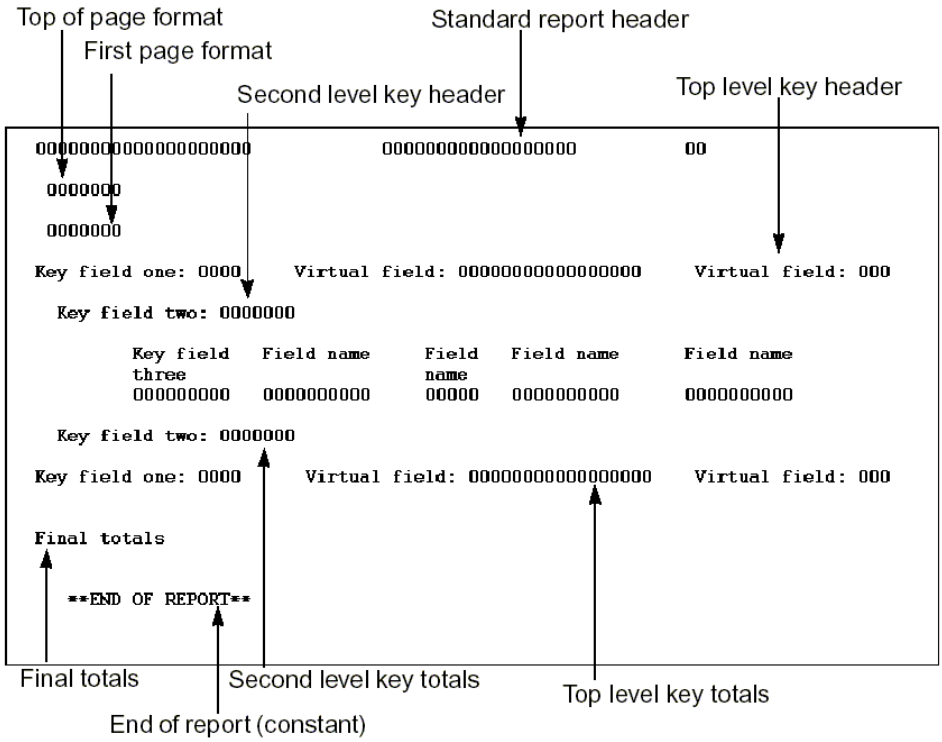
Parameters	Usage	Role	Default	Option
Other fields	Any	/MAP	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Send all error messages	M(YSNDMSG)	Y, N
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Overrides if submitted job	*	N
Distributed file I/O ControL	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter, "Setting Default Option for Your Functions."

The following is an example of a Print File.



For more information on user points, see the chapter, "Modifying Action Diagrams."

PRTOBJ Device Function

The Print Object (PRTOBJ) function defines an embedded report that prints the records from a specified access path at any point within a PRTFIL function. You also can embed PRTOBJ functions within other PRTOBJ functions.

A PRTOBJ function is an internal function, much like the EXCINTFUN, RTVOBJ, CHGOBJ, or (DLTOBJ) functions, in that the high level language source code generated to implement the function is included as source code within the Print File function that calls it.

You can attach a PRTOBJ function to a Retrieval (RTV), a Resequence (RSQ), or a Query (QRY) access path.

Most considerations that affect PRTFIL also affect PRTOBJ.

Device considerations:

- The default device design for a PRTOBJ report design includes first page, detail, and final total formats. The standard header and footer formats and the external features of the report (for example, page size, compiler overrides) are determined by the embedding PRTFIL function.
- If there is more than one key field, CA 2E creates header and total formats for each additional key level. You can drop unwanted formats using the Edit Report Formats panel.

Default logic:

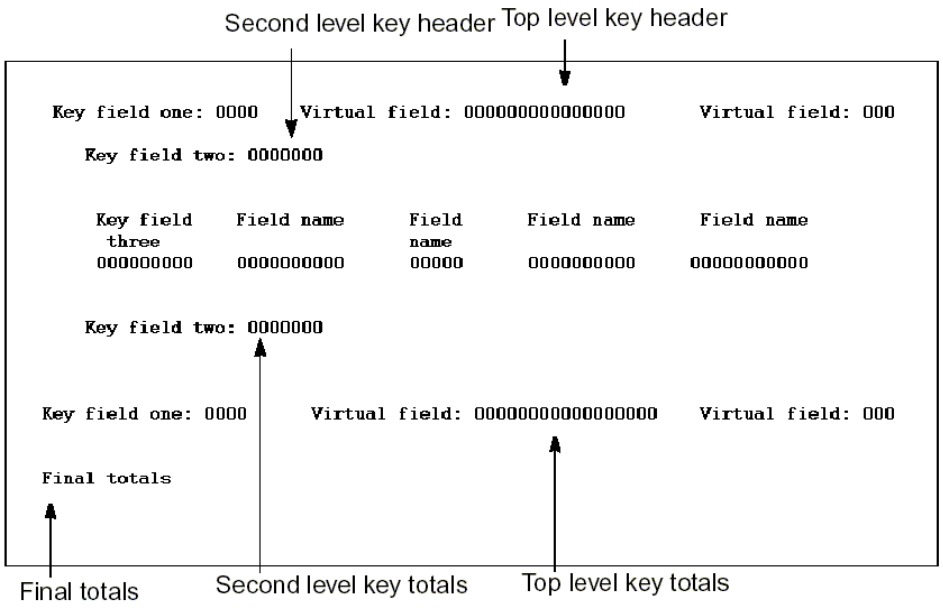
- The PRTOBJ function reads one, several, or many records according to the entry parameters that you specify for it. The same parameter considerations that apply to PRTFIL functions also apply to this function type.
- You can specify totaling with a PRTOBJ function in the same way as for PRTFIL function types (from the CUR context to the NXT context). If you want to return a total to the calling function, you must return it as a parameter

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	I	RST/POS	-	O
Other fields	Any	-/MAP	-	O

There are no function options available for this function.

The following is an example of a Print Object.



RTVMSG Message Function

The Retrieve Message (RTVMSG) function specifies a process that retrieves message text. You can then use the message text to perform any number of processes such as moving a character string (or strings) from a database file into a field. You could also concatenate two discrete character strings from different fields and place them in the same work field or some other text field in another file.

The RTVMSG function is attached to a special CA 2E system file called *Messages.

Default logic:

- CA 2E implements the RTVMSG function via a high-level language program call to a CL program (Y2RVMGC).
- The RTVMSG function returns a message to the calling program (that is, the program that called the RTVMSG function). You would have to declare the field that is to receive the returned message data as an output parameter to the RTVMSG function.

Specifying RTVMSG

To specify a RTVMSG function, define a RTV type message function. Go to the Edit Message Function Details panel (type **Z**). On the message text prompt, specify the specific text along with system parameters that are derived as part of the text string.

RTVOBJ Database Function

The Retrieve Object (RTVOBJ) function specifies a routine to retrieve one or more records from a database file. Processing can be specified for each record read, by modifying the action diagram for the function.

The RTVOBJ function attaches to a Retrieval (RTV), Resequence (RSQ), or Query (QRY) access path, or Physical (PHY). The QRY access path lets you specify virtuals as key fields. There are no function options or device designs available for RTVOBJ.

Note: For more information on PHY, see the section Internal Database Functions and PHY Access Paths.

The RTVOBJ function reads one, all, or a selection of the records or array entries according to the specified entry parameters.

If you need data to be returned from the RTVOBJ to the calling function, you must perform moves within the RTVOBJ user points. The best way to implement this is to use *MOVE ALL from a DB1 context to a PAR context if the record is found. The fields to be passed back to the calling function must be specified as output parameters on the RTVOBJ.

Note: *MOVE ALL only performs moves for fields with matching names.

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Key fields from access path	I	RST/POS	Y	R
Other fields	Any	-	-	O

The following table shows the function options available.

Options	Default Value	Other Values
Share Subroutine	M(YSHRSBR)	N, Y

Effects of Restrictor Parameters

If all the keys of the access path to which a RTVOBJ function attaches are supplied as restrictor parameters, only the record with the given key or keys is read.

If only some of the keys (major keys) are supplied as restrictor parameters, all of the records with the given key are read.

If none of the keys for the access path are supplied as restrictor parameters, all of the records in the access path are read.

Effects of Positioner Parameters

If all the keys of the access path to which the RTVOBJ function attaches are supplied as positioner parameters to the function, only the records with a key value greater than or equal to the given key or keys are read.

If only some of the keys (major keys) are supplied as restrictor parameters, but some or all of the remaining keys are passed as positioner values, only those records with keys equal to the restrictor values and greater than or equal to the positioner values are read.

If none of the keys of the access path are supplied as positioner parameters, all of the records in the access path within the specified restrictor group are read.

For more information on user points, see the chapter, "Modifying Action Diagrams."

Effects of No Parameters

If no keys of the access path are supplied as parameters, there are two possible outcomes:

- All records in the access path are read if the USER: Process Data Record user point contains user logic.
- Only the first record is read if the USER: Process Data Record user point does not contain user logic.

SELRCD Device Function

The Select Record (SELRCD) function defines a program to display the records from a specified file using a subfile. The program enables the end user to select one of the records and the key of the selected record is returned to the calling program.

For each field on the based-on access path, an associated input-capable field is present on the subfile control record. You can use these fields to control how records from the database are displayed. There are three types:

- Restrictor parameters for the subfile (protected key fields).
- Positioner parameters for the subfile (unprotected key fields).
- Selectors (non-key fields). You can specify the operation (Equal to, Contains, Greater than) by which the selection is made, using the Edit Screen Entry Details panel.

The SELRCD function type differs from the Display File function type. Records are similarly displayed as a list or subfile, but the SELRCD function includes processing to return the key values to the panel of the calling function whenever you select an item with a line selection option of 1 (CUA Entry) or **/** (CUA Text) action bar.

You can attach a SELRCD function to a Retrieval (RTV), Resequence (RSQ), or Query (QRY) access path. The QRY access path lets you specify virtuals as key fields.

Effect of parameters:

If you define a partially restricted key to a SELRCD function, the SELRCD function displays a subset of the total number of records. If you do not restrict the keys, you can use them to position the subfile display. To do this, enter the character string required to position the display after the ? in the relevant code field.

Design considerations:

The SELRCD function executes in *SELECT mode only. There is only one display panel for this function type. There is no default update processing for this function.

When you type a ? or user prompt (F4) for a key or foreign key field, CA 2E calls the prompt function assigned to this file-to-file relation. The SELRCD function based on the access path used to validate the value entered becomes the default prompt function.

CA 2E determines the appropriate default prompt function to call for a key or foreign key field as follows:

1. Determine the referencing access path of the relation associated with the field. By default, this is the primary Retrieval access path of the referencing file.
2. Select, by name, the first SELRCD built over that access path.
3. If no SELRCD is found, no call is generated.

The F4 prompt function assignment enables you to select another function to override the function assigned to the access path relationship. This assignment can be made at the access path or function level. You can select any external function except Print File and the function can be based over any access path that is valid for the function type you select. Function level assignments take precedence over access path level assignments.

For more information on instructions for assigning F4 prompt functions, see Editing Device Designs and Building Access Paths in the chapter "Modifying Device Designs," and Changing a Referenced Access Path in the chapter "Modifying Access Paths."

The following table shows the parameters available.

Parameters	Usage	Role	Default	Option
Return code	B	–	Y	R

Parameters	Usage	Role	Default	Option
Key fields	B	MAP	Y	R
Other fields	B	RST	–	O
non-key fields	I	MAP	–	O
Other fields	Any	-/MAP	–	O

If there are output or both parameters, these are passed back automatically if the fields are present in the subfile record.

The following table shows the function options available.

Options	Default Value	Other Values
Subfile end	M(YSFLEND)	P, T
Send all error messages	M(YSNDMSG)	Y, N
If action bar, what type?	M(YABRNPT)	A, D
Commit control	N(*NONE)	M(*MASTER), S(*SLAVE)
Generate error routine	M(YERRRTN)	Y, N
Reclaim resources	N	Y
Closedown program	Y	N
Copy back messages	M(YCPYMSG)	Y, N
Generation mode	A	D, S, M
Screen text constants	M(YPMTGEN)	L, I
Generate help	M(YGENHLP)	Y, N, O
Help type for NPT	M(YNPTHLP)	T, U
Workstation implementation	M(YWSNGEN)	N(NPT), G(GUI), J(JAVA), V(VB)
Distributed file I/O Control	M(YDSTFIO)	S, U, N

For more information on function options, see the chapter, "Setting Default Options for Your Functions."

The following is an example of a SELRCD.

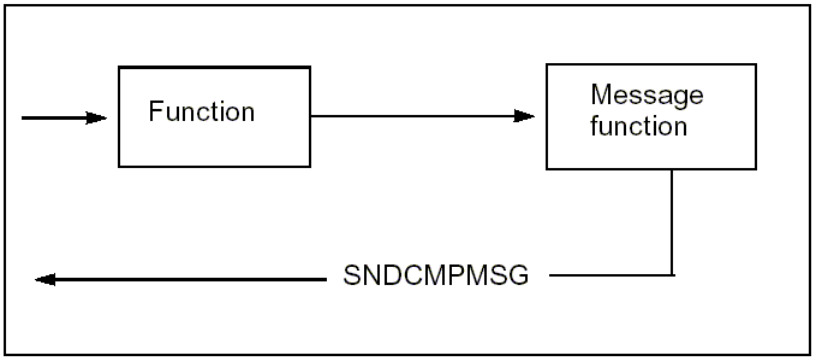
```
                                Select Order
Order
code
_____

Type options, press Enter.
1=Select

Opt  Order   Customer  Customer name          Employee name
     code    code
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000
_    000000  000000    00000000000000000000  00000000000000000000000000 +

F3=Exit   F4=Prompt
```

For more information on user points, see the chapter "Modifying Action Diagrams."

SNDCMPMSG Message Function

The Send Completion Message (SNDCMPMSG) function specifies that a completion message is sent to the function that called a function. A completion message indicates completion of a particular task.

The SNDCMPMSG function causes a message to be returned to the message queue of the program that occupies the previous higher position in the invocation stack of the program that invokes the SNDCMPMSG function.



The SNDCMPMSG function is attached to a special CA 2E system file called *Messages.

CA 2E implements the SNDCMPMSG function using a shipped user program called Y2SNMGC.

Example

In this example, you have a Display File function that calls a separate Print File function to print out some details. You might then send a completion message from the Print File function to indicate that the print is complete. This could be done by modifying the action diagram of the Print File function as follows:

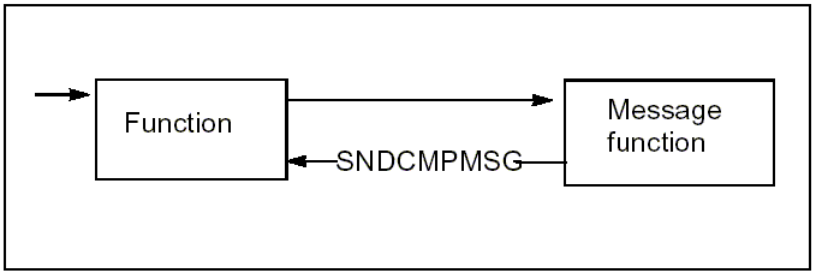
```
>  USER: Process end of report
:—
:Send completion message - 'Print of details complete'          <<<
'—
```

The message then appears automatically on the message subfile of the Display File function.

SNDERRMSG Message Function

The CA 2E Send Error Message (SNDERRMSG) function is used to send an error message to the message queue of the calling program. An error message indicates that an error occurred arising from validation of user-entered data.

The SNDERRMSG function causes a message to be sent to the program message queue of the program that calls the SNDERRMSG function. All standard functions have message queue subfiles and display messages at the bottom of the panel.



The SNDERRMSG function is attached to a CA 2E shipped file called *Messages.

When you call a SNDERRMSG function, you can use any input-capable fields that you defined as input parameters, as message substitution data. These parameter fields do not need to appear in the text of the message.

When you call a SNDERRMSG function, input-capable fields on your function's device design, which you defined as Input or Neither parameters to the SNDERRMSG function, appear highlighted and in reverse image unless Flag Error is set to N on the Edit Function Parameter Details panel for the device function. The cursor is positioned at the first error field, unless overridden by the *SET CURSOR function. The program does not proceed past the validation subroutine until you correct the error.

When you call a SNDERRMSG function, the program encounters more than one error, all input-capable fields in error are highlighted. Depending on what you specified for the function option Send All Error Message, the program sends only the first error or all errors.

Consider the following example. If you have a device function that requires a calculated total value to be checked against an entered total value, and an error message sent if these totals do not agree, you would do this by:

Defining an error message.

Modifying the action diagram of the device function to call the SNDERRMSG function in the appropriate circumstances as shown next.

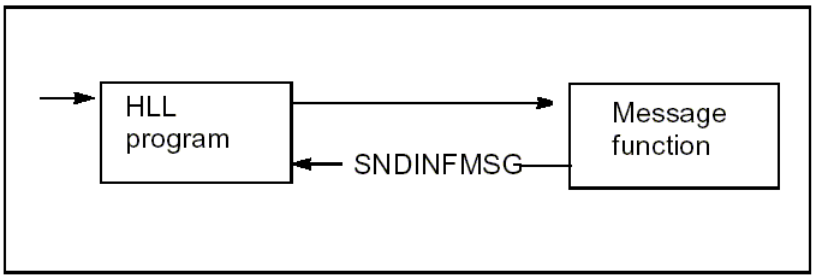
```
> USER: Validate totals
: - -
:   .-CASE                                                              <<<
:   | -CTL. Calculated total NE CTL. Entered total                      <<<
:   | Send error message - do not agree'                                <<<
:   '-ENDCASE
'- -
```

The device function repeatedly redisplays the fields until the error is corrected.

SNDINFMSG Message Function

The Send Information Message (SNDINFMSG) function is used to send an information message to the message queue of the calling program. An information message informs or provides information concerning a particular task.

The SNDINFMSG function causes a message to be sent to the program message queue of the program that invokes the SNDINFMSG function.



The SNDINFMSG function is attached to a CA 2E shipped file called *Messages.

Consider the following example. If you have a CA 2E device function that calculates a control total as part of the validation and the calculated total value differs from an entered total value, you may want to send an information message as a warning that the end user can choose to ignore.

To do this, you first define an information message in the normal way and modify the action diagram of the device function as follows:

```
>   USER: Validate totals
:—
: .CASE                                                                  <<<
: |  -CTL. Calculated total NE CTL. Entered total                       <<<
: |      Send  information message - 'Totals disagree'                   <<<
: 'ENDCASE
'_
```

SNDSTSMSG Message Function

The Send Status Message (SNDSTSMSG) function is used to send a status message to the message queue of a calling function. A status message provides information concerning the progress of a long-running task.

The SNDSTSMSG function causes a message to be returned to the program message queue of the job that invokes the SNDSTSMSG function. The SNDSTSMSG function type is only valid for interactive jobs and causes the message to display at the base of the panel during a long-running interactive job.

The SNDSTSMSG function is attached to a CA 2E shipped file called *Messages.

Example

You might have a function to execute End of Year processing that is a long-running process. Before starting the main body of the function, you could send a status message to indicate that it is in progress; for example:

```
> USER:
:—
: Send status message - 'End of year processing running'                <<<
'—
```

SUM Function Field

The Sum function field is a special field usage used within functions to contain the result of a computation of the sum of values of another field.

Function fields of type SUM must be numeric. If the function field is defined as a reference (REF) field that is based on another field, the SUM field contains a summation of the values of the referenced field. The referenced field must also be numeric.

Function fields of type SUM always have two parameters:

- A result parameter: This is the actual field itself containing the results of a summation. You must place the field on a totaling format or subfile control format of the Display or Edit Transaction that calls the SUM function field.
- An input parameter: This is the field whose sum will be calculated. This field must exist on the detail format of the function using the SUM function field.

The following are examples of Sum fields:

- Total order value: the sum of order lines
- Total warehouse space: sum of location space

USR Function Field

The USR (User) field usage is reserved for any work fields that you need on a device design or in an action diagram. A USR function field can be input-capable or output only. USR function fields are usually defined as REF fields to existing database fields.

There is no default processing for USR function fields. You must initialize, validate, or specify any special processing for the field. However, CA 2E performs basic field type validation such as date validation if the field is a DTE type field.

The following are examples of User fields:

- Command request strings
- Menu options
- Work fields used on the panels or in the function's action diagrams

Default Prototype Functions

The Default prototype function is available for *Template functions. It is available only for functions created over the *Template file and is available for all function types (CHGOBJ, EDTFIL, and so on). This function facilitates the use of *Template functions. When new functions are created, the use of a specific *Template function is enforced.

Note: In previous versions, users had to select *Template functions (by using F21) explicitly while creating new functions. There was no enforcement of *Template usage.

If a function x based on the *Template file that has the Default prototype function function option set to Y, then any function of x type subsequently created over any user-defined (non-system) file in the model automatically uses function x as their prototype, rather than using the system default.

Note: New functions created over the *Template file always use the system default at the time of creation.

For all functions based over the *Template file, only one function of each type can have the Default prototype function function option set to Y. Any user with DSNR authority can change the Default prototype function option.

Chapter 4: ILE Programming

2E supports ILE programming in the form of 2 HLL codes, RP4 and CBI. Functions using the RP4 HLL code have code generated by the RPGIV generator; CBI functions generate CBL ILE code using the standard shipped COBOL generator

This section contains the following topics:

Choosing RPGIV as the Default Language

To create a new model with RPGIV as the default language, use the YCRTMDLLIB command as follows:

```
YCRTMDLLIB…HLLGEN(*RPGIV)
```

This value becomes the model value YHLLGEN. In addition, the default binding directory YBNDDIR is created in the generation library.

To make RPGIV the HLL generator for an existing function, change the value on the Edit Function Details panel to RP4.

ILE Features That Affect CA

The IBM integrated Language Environment (ILE) includes many enhancements and changes over the Original Program Model (OPM) that controls and supports RPG and COBOL programs.

This section contains brief descriptions of the ILE features that the CA 2E RPGIV ILE generator uses for program creation and program calling.

Note: The RPGIV generator includes processing that can use such ILE features as bound (static) calls, activation groups, binding directories, and more. These features provide functionality that is not available for RPG or COBOL programs. A full understanding of ILE is therefore necessary for using the RPGIV generator. See the documentation from IBM or another source.

Program Creation

In OPM, program creation consists of compiling source code into runnable program objects (*PGM). A program object is created from a single source member using the Create RPG Program command (CRTRPGPGM).

By contrast, in ILE, program creation consists of:

- Compiling source code into nonrunnable module objects (*MODULE)
- Binding (combining) one or more modules into a runnable program object (*PGM)

One way to create an RPGIV program object is the same way you create an RPG program in the OPM framework: using the CRTBNDRPG command. This command creates a temporary module that is bound into a program object and later deleted. This is the quickest and simplest way to create an ILE program.

Another way to create an RPGIV program object is using separate commands for compilation and binding. In this two-step process, you create a module object with the Create RPG Module command (CRTRPGMOD). This command compiles the source statements into a nonrunnable module object, which must be bound into a program object with the Create Program command (CRTPGM).

ILE also lets you bind other objects using a binding directory. A binding directory is essentially a "list" of modules that may be needed when the program runs. When the CRTBNDRPG command specifies a binding directory, the compiler or binder searches the binding directory to see if the program being compiled accesses any modules in the directory. If it does, the compiler or binder binds them to the program. A binding directory can reduce program size because the modules or service programs in a binding directory are used only when needed. For more information about the binding directory, see the section The YBNDDIR Model Value.

CA 2E lets you define a function as either a module or a program. During the source generation and compilation steps, the RPGIV ILE generator ensures that references to bound objects are correct and creates either a program object (*PGM) or a module object (*MODULE). Created modules can be bound to created programs during compilation, either explicitly or through the default CA 2E binding directory.

Program Calling

In ILE, you can write applications in which ILE RPG/400 programs and OPM RPG/400 programs interrelate by using the traditional dynamic program call. The calling program specifies the name of the called program on a CALL statement. The name of the called program is resolved to an address at run time, just before the calling program passes control to the called program. The program name may be known to the program only when the call is made (perhaps if the program to be called is a variable value). Because of this, the dynamic call uses considerable system resources, and repeated dynamic calls can reduce the performance of the calling program.

You can also write ILE applications that interrelate with faster static calls. Static calls are calls between procedures. A procedure is a self-contained set of code that performs a task and then returns to the caller. An ILE RPG/400 module consists of an optional main procedure followed by zero or more subprocedures. Because the procedure names are resolved at bind time (that is, when you create the program), static calls are faster than dynamic calls.

Example: The CA 2E generator uses static calls where possible. Suppose a model contains the following external functions **Function PGM**—generated in RPGIV, compiled as a program object (*PGM)

Function MOD—generated in RPGIV, compiled as a module object (*MODULE)

Function MOD2—generated in RPGIV, compiled as a module object (*MODULE)

Function RPG—generated in RPG or COBOL, compiled as a program object (*PGM)

In a model with the functions just listed, the following call situations occur:

1. Function PGM calls Function MOD, using the Call Bound Procedure statement (CALLB).
2. Function MOD calls Function MOD2, using the CALLB statement.
3. Function RPG calls Function PGM, using the Call Program statement (CALL).
4. Function PGM calls Function RPG, using the CALL statement.
5. Function RPG calls Function MOD, using the CALL statement. In this case, however, the call fails because RPG programs cannot call module objects using a dynamic call.

Generating RPGIV Source

The process to invoke source generation is the same as for RPG:

- Enter a G next to the function to generate the source interactively.
- Enter a J next to the function to generate the source and compile the object in batch.

After the source is generated, you can view or edit it by entering an E next to the function. The source file is QRPGLESRC and the source type is RPGLE.

Control (H) Specifications

The RPG generator uses the contents of the YRPGHSP model value as the Control (H) specification for the generated RPG source. The RPGIV generator, however, uses the contents of two separate model values:

YRP4HSP—Control (H) specification for objects of type *PGM

YRP4HS2—Control (H) specification for objects of type *MODULE

In addition, and unlike the RPG generator, you can add extra H lines using user source; see the section RPGIV User Source. This is because the H-specification is keyword-based and can take more than 80 characters.

Note: You can change the model values YRP4HSP and YRP4HS2 with YCHGMDLVAL, but they are too long to be displayed on the Display Model Values panel.

Compiling RPGIV Source

As stated earlier, you can compile generated RPGIV source into a program object (*PGM) with CRTBNDRPG or a module object (*MODULE) with CRTRPGMOD. To accommodate this choice between object types, the Edit Function Details panel has two new options, O and T:

```
SEL: E-STRSEU, O-Compiler Overrides, T-ILE Compilation Type (*PGM/*MODULE)
```

Option O

The O option controls the compiler overrides. Use this option if you want additional binding directories (to use IBM APIs, for example). Because the CRTBNDRPG and CRTRPGMOD commands have different default values, changing the object type with the T option deletes any compiler overrides for the previous object type.

Option T

The T option toggles between PGM (*PGM) and MOD (*MODULE) as the object type created when the source is generated and compiled. This option is available only for a target HLL that is ILE compatible, like RPGIV. The current object type is shown at the left of the subfile line. During generation, the change is limited to the compiler overrides in the source (the Z* lines). Here are more details about the compile options:

- **PGM—**The IBM Create Bound RPG Program command (CRTBNDRPG) compiles the generated source into a program object (*PGM). The command creates a temporary module, binds that module into a program, and then deletes the temporary module.

  The defaults for this command are in the *CRTBNDRPG message in the *Messages file:

CRTBNDRPG PGM(&2/&1) SRCFILE(&3/QRPGLESRC) DFTACTGRP(*NO) BNDDIR(&YBNDDIR) DBGVIEW(*SOURCE) CVTOPT(*DATETIME) ACTGRP(*CALLER) OPTIMIZE(*BASIC)

For details about the BNDDIR parameter value, see the section The

- **MOD—**The IBM Create RPG Module command (CRTRPGMOD) compiles the generated source into a module object (*MODULE). You must then bind that module into an ILE program, possibly with other modules.

  The defaults for this command are in the *CRTRPGMOD message in the *Messages file:

CRTRPGMOD MODULE(&2/&1) SRCFILE(&3/QRPGLESRC) DBGVIEW(*SOURCE) CVTOPT(*DATETIME)

RPGIV User Source

Functions generated with source type RP4 should include only user source of the type RP4. This user source lets you take advantage of some features of the RPGIV language that are not currently available in the CA 2E model generated source. An example is the use of pointer variables.

RPGIV user source must reside in QRPGLESRC. The rules for naming parameter variables in RPGIV user source are the same as for RPGIII user source. You can include all RPGLE specification types in the source member. CA 2E sorts the specification types into their appropriate positions within a program.

CA 2E codes the source in the normal order for RPGLE specifications, which is:

1. H specification

   Any H specification lines that you add are placed after the default H specification lines generated for the owning function source. The H specifications are taken from the YRP4HSP and YRP4HS2 model values.

2. F specifications
3. D specifications
4. I specifications
5. C specifications
6. O specifications
7. P specifications, including any contained D and C specifications.
8. Arrays

CA 2E treats the first RPG calculation specifications, which are not part of a subroutine, as the instance code. For repeated calls to an EXCUSRSRC function, CA 2E generates the code on every call to the function at the point indicated by the action diagram.

CA 2E automatically inserts into the ZZINIT subroutine any C specifications that follow a subroutine but are not part of a subroutine. The order of specification is:

1. Instance (mainline) code C specifications
2. C specifications for subroutines called by instance code
3. Initialization C specifications

Note: In version 7.0, parameter fields (fields prefixed with #O, #I and #B) are recognized only in the Factor One, Factor Two, and Result positions of the RPGLE calculation specifications that are part of the instance code. They can be in upper, lower, or mixed case. They are not currently recognized in free-format expressions.

The following example shows an RPGIV function called Get Key that is defined into EXCUSRSRC Function another CA 2E standard function. The EXCUSRSRC function has three parameters:

IOB	Parameter	GEN name
I	Job date	JDT
I	User name	USR
O	Encoded file key	ABVN

The sample shows user source coded as a prototyped procedure with several distinct sections of code:

- The procedure prototype (D specifications)
- Inline code (to execute the procedure)
- Procedure code (including more C and D specifications)

```
 * PROCEDURE PROTOTYPE                                                  ) Inserted into
D Get_Key              PR             10                               ) inline D-specs
D Job_Date                            7       0 VALUE                  )
D Job_User                            10        VALUE                  )
 *
 * USER_DEFINED FIELDS
D Key_Date             S              7       0
D Key_User             S              10
D Enc_Val              S              10
 *
 * IN-LINE PROCESSING
 *
 * Use job date and user name as input parameters to Get_Key proc.
C              Move           #ijdt          Key_Date                 )
C              Move           #iusr          Key_User                 ) Inserted into
C              Eval           Enc_Val = Get_Key(Key_Date:Key_User)     ) inline C-specs
 *                                                                     )
 * Return encoded value as parameter from user source function        )
C              Move           Enc_Val        #oadvn                   )
 *                                                                     )
 *
 * PROCEDURE DEFINITION                                               )
```

```
PGet_Key              B                                                 )
 *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *    )
DGet_Key              PI                    10                          )
D Job_Date                                  7       0 VALUE             )
D Job_User                                  10        VALUE             ) Inserted into
 *                                                                      ) source after
D Job_Date_Ptr        S                     *                           ) all C-specs and
D Job_Date_Char       S                     7         Based(Job_Date_Ptr) ) all system-
D File_Key            S                     10                          ) generated
 *                                                                      ) procedure code
C             Move    Job_Date       Job_Date_Char                      )
C             Eval    File_Key =     %Subst(Job_User:3:2) +             )
C                                    %Subst(Job_Date_Char:3:3) +        )
C                                    %Subst(Job_User:6:2) +             )
C                                    %Subst(Job_Date_Char:1:3)          )
C             Return  File_Key                                          )
P                     E                                                 )
```

The user source procedure code does not need to conform to the CA 2E model naming conventions. Field names used only in the procedure can have the full 14-characters that the RPGIV language allows. Also, #I and #O must be within C specification lines and not within free-format expressions like the EVAL statement.

Model Value YRP4SGN

The RPGIV generator includes some source generation options that you can set at a model level. These options are in the new model value YRP4SGN in a data area called YRP4SGNRFA (RPGIV source generation options). YRP4SGNRFA is a 16-character data area, and a copy is created in each version 7.0 model library. The 16 characters are:

- **Character 1—**This character determines whether the RPGIV source is in upper case, lowercase or mixed case:
  - U—upper case (default)
  - L—lowercase
  - M—mixed case (first letter in upper case)
  - If Character 1 is L or M, subroutine names and internal TAG labels are upper case.

- **Character 2**—This character determines the color of the comments in the generated RPGIV source:
  - G—green (default)
  - W—white
  - R—red
  - P—pink
  - B—blue
- **Characters 3 - 16**—These are not used in version 7.0, but are available for future use.

The default value, which is UG, means that the RPGIV source is upper case with green comments. If you change the value to MW, the source would be mixed case with white comments. Therefore, RPG would generate a field name as WUABVN, but RPGIV generates it as Wuabvn. Likewise, op-codes such as EVAL, IF, and SETOFF are Eval, If, and Setoff.

RPGIV Generator Notes

The RPGIV ILE generator lets you create ILE programs and modules from generated RPGIV source. However, for version 7.0 of CA 2E, note the following details and limitations:

- Unlike the RPG and COBOL generators, the RPGIV generator is available free of charge to any customer who already has a fully licensed CA 2E model. For administration purposes, however, it is licensed separately, and you must specifically request the license.
- Although the RPGIV generator options can create both *PGM and *MODULE objects, the CRTPGM and CRTSRVPGM commands are not currently available to create an ILE program or service program.
- The CRTBNDRPG and CRTRPGMOD default commands are held as the *CRTBNDPRG and *CRTRPGMOD messages Y2U1022 and Y2U1024 respectively in the *Messages file. If you wish, you can change the default parameters or add new parameters.

Service Program Design and Generation

In the CA 2E model, a function type called Service Program (*SRVPGM) is defined. Service Programs can be copied, deleted, renamed, and so on, just like any other function. Service Programs have no parameters or action diagram, and they can be defined over any file. A Service Program has an associated source member, whose name is automatically allocated, but which can be changed. The source member name will be used for the final *SRVPGM object name.

Once a Service Program is created, a developer can, through a series of screens, define which existing module functions should be bound into the Service Program. Additionally, they can define one or more external (non-2E) *MODULE objects to be bound into the Service Program. For each bound module, the developer can specify which exported procedures from the module should also be exported from the Service Program.

When the Service Program is generated, a source member is created which includes both the export list, as well as several compile preprocessor directives which define how the *SRVPGM object should be created.

Service Program Overview

Within the CA 2E product, external functions can be defined with an object attribute of either PGM or a MOD. When compiled, these functions become one of two i OS object types-respectively, a *PGM (executable program) or a *MODULE (non-executable module).

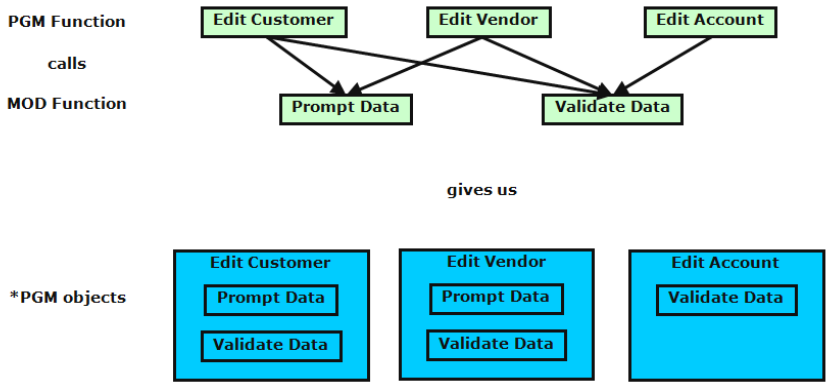
A *PGM object can be directly called, either from a command-line or from another program (assuming the correct parameters are passed). By contrast, a *MODULE object must first be 'bound' into an executable program.

Earlier releases of CA 2E provided the following ILE-specific functionality:

- An external function could be given an attribute of MOD (instead of PGM). This function would compile into a i OS *MODULE rather than a *PGM.
- All MOD functions are automatically added to the 2E model's default binding directory YBNDDIR during the compilation process.
- The default compilation command for ILE programs forces the compiler to search YBNDDIR for called bound modules.
- Calls to a MOD function would be generated (in RPG) as a CALLB (call-bound) rather than a CALL.

Note: Developers could then define certain external functions (typically ones that would never be called from a command line, such as SELRCD's) as modules. Any programs that called these functions (EDTFIL's, etc.) would include a copy of the module object. In ILE terms, this is known as *bind-by-copy*.

For example, the following diagram shows the use of three PGM functions calling two MOD functions and the resulting *PGM objects containing copies of the *MODULE objects:



This has a number of benefits:

- Fewer *PGM objects required at runtime can simplify application deployment.
- Calls to bound modules are typically quicker than calls to other programs, due to significantly reduced object pointer resolution overhead.

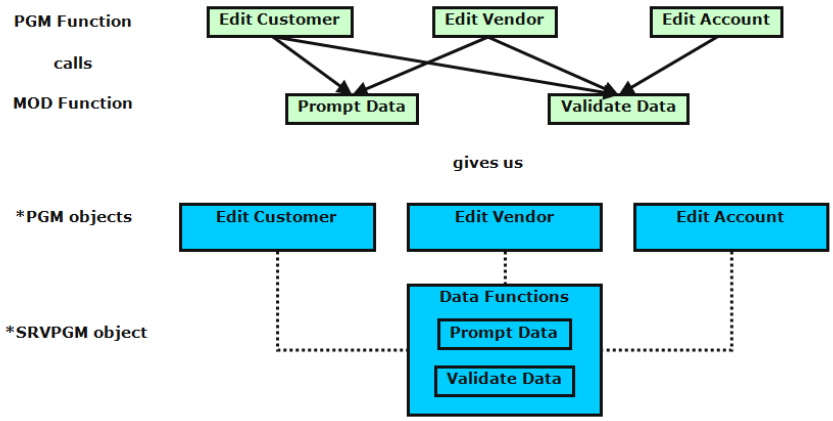
However, if a module is bound into a program and the code within the module function needs to be changed, the *MODULE must be recompiled and then any programs that bind that module must be changed to include the new copy of the module-typically this would be done by regenerating/recompiled the PGM function in a 2E model, although the i OS UPDPGM command can be run from outside the model environment.

Note: Although there are fewer *PGM objects, they are larger than if they did not contain a copy of the module.

Service Program Functions

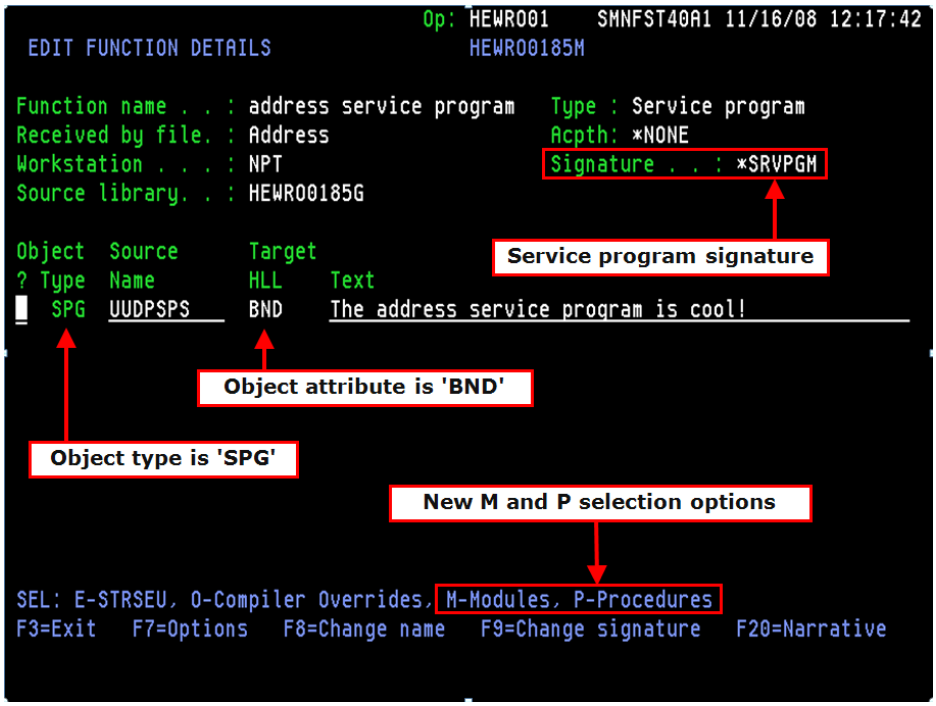
The new Service Program Support allows developers to create service program functions, which are functions that equate to i OS *SRVPGM objects and can be thought of as *containers* for one or more *MODULE functions. As with modules, you cannot directly call service programs. However, when a program function is compiled into a *PGM object, if it finds the service program (which contains the module) in the binding directory before it finds the module itself, it performs a *bind-by-reference*, wherein it simply contains a reference (link) to the module object in the service program and does not itself contain a copy of the module object. At runtime, the program resolves (calls) to the copy of the module contained in the service program (so the service program needs to be available).

Using service programs in this manner provides the benefit of improved performance, due to one-time object resolution, but ensures that a change in any of the modules contained in the service program does not require a change in the calling program, as detailed in the following diagram.



Edit Function Details Panel

This existing panel displays when you take option *Z* (Details) against a function. For a Service Program, this screen contains certain Service Program-specific fields, subfile options, and function keys, as you can see in the following example:



The *Signature* field displays the current signature for the Service Program. It can take the following values:

- *SRVPGM - The signature is the *SRVPGM object name
- *GEN - The signature is system-generated
- User-defined - the signature is a user-specified 16-byte string

Note: The signature is determined at generation time (or in the case of *GEN, at compile-time).

You can take option M (Modules), to display the list of modules currently bound into the Service Program (and to add more modules if you authority).

You can also take option *P* (Procedures) against the Service Program, to display the list of procedures which will be exported from the *SRVPGM object (and, if you have sufficient authority, to reorder that list).

Use function key F9 to make the Signature field input-capable

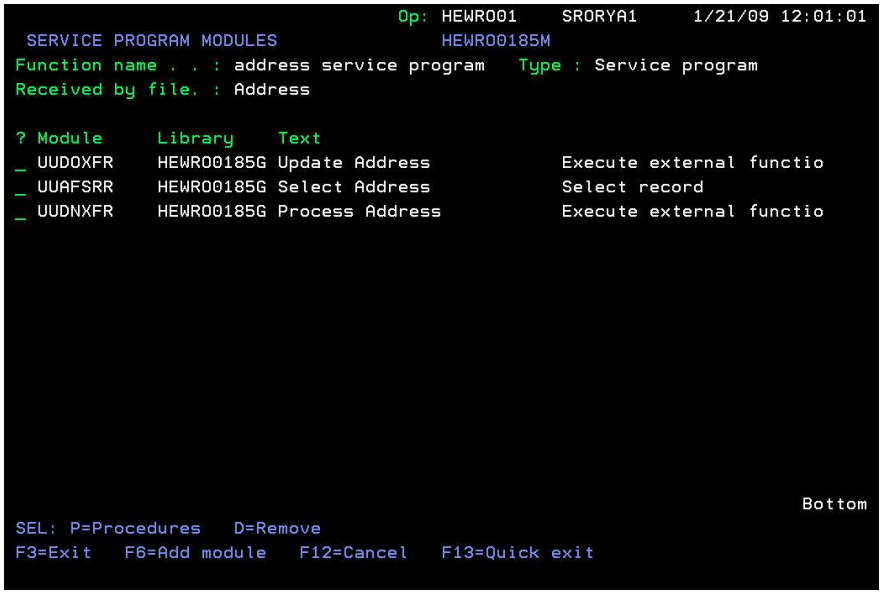
There is one function option for a Service Program function - *Add to Binding Directory*. This determines whether or not this service program should automatically be added to the model's default binding directory. Whether set to Y or N, all constituent modules will be removed from the default binding directory when the service program is created.

Setting this to N can be useful where you create multiple service programs that contain one or more of the same modules, which therefore export the same procedures, since this would give rise to errors when subsequently compiling programs.

Adding Modules and Procedures

The Service Program Modules panel is called when option M is taken from the EDIT FUNCTION DETAILS panel (above).

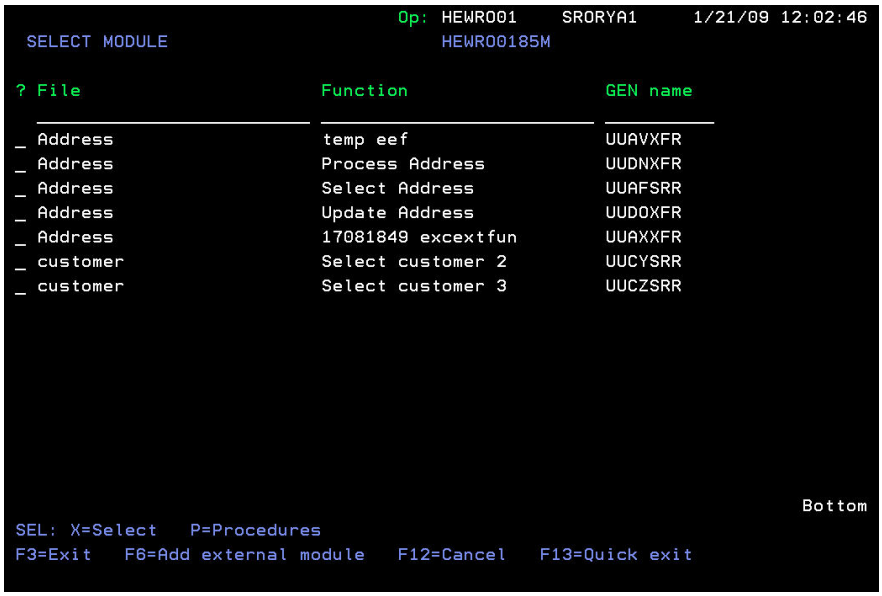
Service Program Modules



From this screen, you can see which modules are currently bound into the Service Program. This list may include both 2E modules and external (non-2E) modules.

You can also press F6 to go to the SELECT MODULE screen

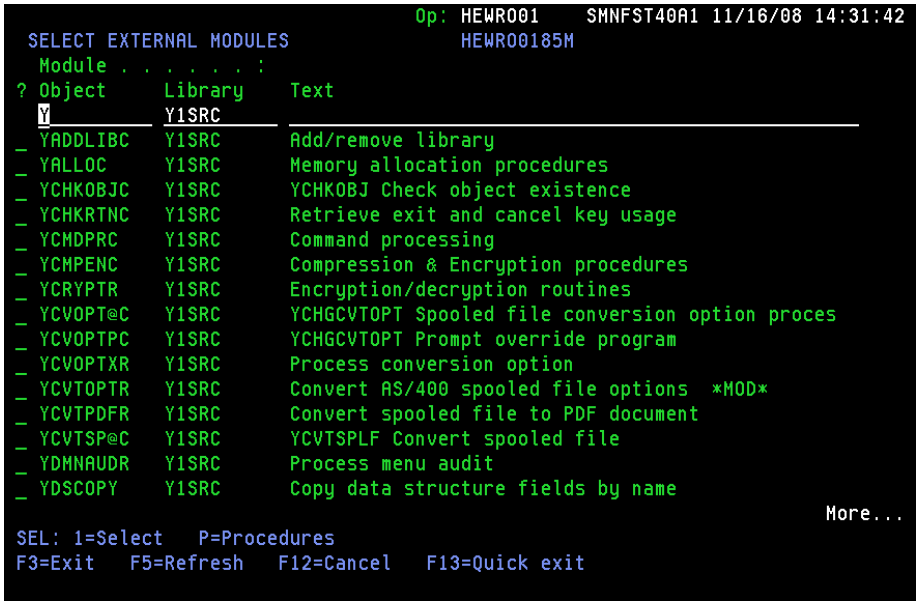
Select Module



You can select one or more modules to bind into the Service Program, by using subfile option X (displayed) or 1 (not displayed, but active anyway). This automatically makes all the procedures from the module exported from the Service Program. Alternatively, you can take option P against a module and select which procedures they would like to export from the Service Program.

You can also press F6 to display the SELECT EXTERNAL MODULE screen

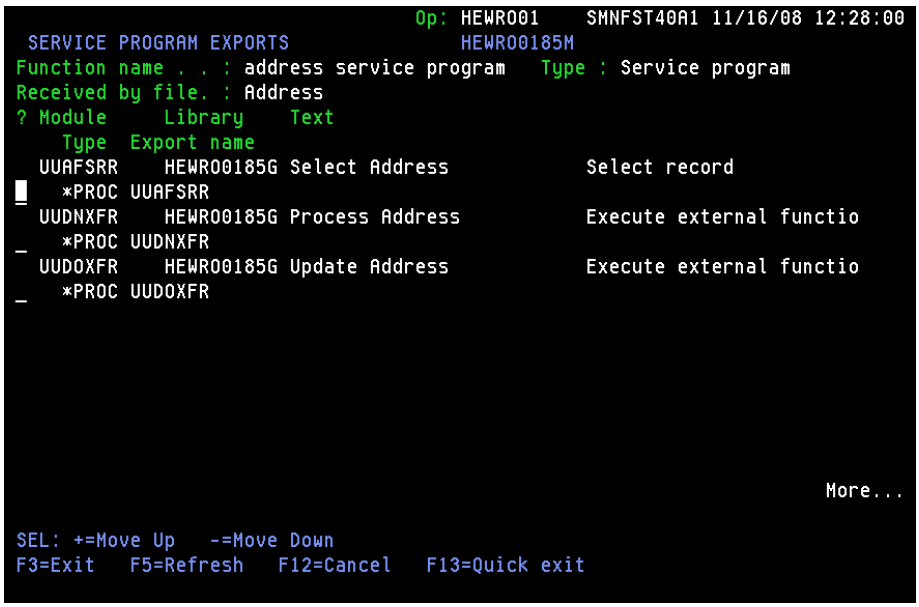
Select External Module



You can select one or more external (non-2E) modules to bind into the Service Program. The initial screen is displayed empty, and you can specify subfile control criteria so sub-select the modules to display. As with the SELECT MODULE screen, option P is available to sub-select procedures from within the module to export from the Service Program.

Service Program Exports

The Service Program Exports panel is displayed when option P is taken from the EDIT FUNCTION DETAILS panel:



This allows you to display the list of procedures which are exported from the *SRVPGM object, and if necessary, re-order them.

Note: Although this is only a requirement where, due to modules being removed and added, the list may have changed-typically, once a Service Program has been generated and compiled into a *SRVPGM object, this list would never be changed).

Service Program Generation Mode

When option G or J is taken against a Service Program, the generation program creates/updates the source member (in file QSRVSRC) in the 2E model generation library (*GENLIB). This source member contains both the export list associated with the *SRVPGM object, as well as several compile preprocessor directives which will be used by the CA 2E Toolkit to actually create the *SRVPGM object. These include the following (in order):

1. The actual CRTSRVPGM command, specifying the bound modules and also specifying the same source member as the export list definition member
2. An ADDBNDDIRE command, to add the*SRVPGM object to the YBNDDIR binding directory (Only if function option Add to Binding Directory is set to Y)

3. Multiple RMVBNDDIRE commands (one for each 2E module bound into the Service P), to remove that module from the YBNDDIR binding directory.
4. When you compile a Service Program function using the YSBMMDLCRT command, the 2E model generation/compile processing executes the YEXCOVR command against the source member. This command invokes the Toolkit compile preprocessor, which processes the compile directives in the source member.

An example of an Service Program source member is as follows:

```
/*Y: CRTSRVPGM SRVPGM(HEWRO0185G/UUDPSPS) +                          */
/*Y: MODULE(HEWRO0185G/UUAFSRR HEWRO0185G/UUDOXFR) +                 */
/*Y: EXPORT(*SRCFILE) SRCFILE(HEWRO0185G/QSRVSRC) OPTION(*DUPVAR +   */
/*Y: *DUPPROC *NOWARN) BNDDIR(QC2LE YBNDDIR) TEXT('The address +     */
/*Y: service program is cool!')                                      */

/*Y: ADDBNDDIRE BNDDIR(HEWRO0185G/YBNDDIR) OBJ((HEWRO0185G/UUDPSPS + */
/*Y:  *SRVPGM)) POSITION(*FIRST)                                     */

/*Y: RMVBNDDIRE BNDDIR(HEWRO0185G/YBNDDIR) OBJ((*LIBL/UUAFSRR +      */
/*Y: *MODULE))                                                       */
/*Y: RMVBNDDIRE BNDDIR(HEWRO0185G/YBNDDIR) OBJ((*LIBL/UUDOXFR +      */
/*Y: *MODULE))                                                       */

            STRPGMEXP  PGMLVL(*CURRENT) SIGNATURE(*GEN)

            EXPORT     SYMBOL('UUAFSRR')
            EXPORT     SYMBOL('UUDOXFR')

            ENDPGMEXP
```

Processing this source member results in the UUDPSPS service program being created in library HEWRO0185G and being added to the top of the YBNDDIR binding directory, with *MODULE objects UUAFSRR and UUDOXFR being removed from the same binding directory.

Note: The shipped defaults for the CRTSRVPGM, ADDBNDDIRE, and RMVBNDDIRE commands can be customized by doing the following:

1. CRTSRVPGM - the defaults for this command are held in a shipped model message, *CRTSRVPGM, with message id Y2U1033:

   ```
   *CRTSRVPGM                EXC    Y2U1033    Y2USRMSG
   ```

2. ADDBNDDIRE - the defaults for this command are held in message Y2R0130 in Y2MSG.
3. RMVBNDDIRE - the defaults for this command are held in message Y2R0137 in Y2MSG.

The YBNDDIR Model Value

The new model value YBNDDIR specifies a binding directory that can resolve the location of any previously compiled RPGIV modules. Use this model value while compiling RPGIV programs with the CRTBNDRPG command. The default CRTBNDRPG command contains the following parameter:

```
BNDDIR(&YBNDDIR)
```

During the pre-compiler process, the value &YBNDDIR is replaced with the value specified for the YBNDDIR model value, even if the value specified in the YBNDDIR model value is *NONE.

Specifying *NONE

A value of *NONE for the YBNDDIR model value causes the following:

- No static binding takes place during RPGIV program compilation. RPGIV programs use:

CRTBNDRPG...BNDDIR(*NONE)

- RPGIV modules must be explicitly bound to a generated RPGIV program. Change the compile overrides for that RPGIV program function by taking option O from the Edit Function Details panel. In addition, specify a binding directory that already has an entry for each module.

Specifying a Value Other Than *NONE

A value other than *NONE for the YBNDDIR model value causes the following:

- The compiler attempts to bind any called modules by checking the binding directory for each called module. RPGIV programs use:

CRTBNDRPG...BNDDIR(binding-directory)

- RPGIV modules generated with the CRTRPGMOD command have the following Y* (pre-compiler) line inserted before the Z* (compile parameter) lines:

Y* ADDBNDDIRE BNDDIR(binding-directory) OBJ((source-member *MODULE))

This adds an entry for the *MODULE function to the specified binding directory at compile time. Any external functions compiled later with CRTBNDRPG…BNDDIR that call a *MODULE function use the Call Bound Procedure statement (CALLB). This improves performance. However, make sure that a called module is compiled before the program that calls it, otherwise the compilation will fail because no entry for the module will be in the binding directory. If several objects are compiled at once, the job list processing ensures this.

Note: If you use YCHGMDLVAL to change YBNDDIR to a value other than *NONE, the command processor determines whether a binding directory of that name already exists in the generation library. If it does not, a directory is created with PUBLIC(*CHANGE) authority.

Chapter 5: Web Service Creation

This chapter describes the mechanism to expose CA 2E server-side programs via web services, and to model this exposure.

The runtime functional deliverable is an automatically created and deployed Web Service(s); its operations invoke 2E server-side ILE service programs.

The feature creates a Web Service containing one-to-multiple operations, where each operation corresponds to a single procedure in a module within a 2E ILE Service Program. Note that CA 2E Service programs can also contain modules developed outside of a 2E model.

This section contains the following topics:

Approach

IBM's i 6.1 and higher provides a Web Services Server.

IBM states, "The Web services server provides a convenient way to externalize existing programs running i OS®, such as RPG and COBOL programs, as Web services."

The IBM Web Administration Interface provides a web-based, wizard like approach to creating and deploying a Web Service that can invoke an RPG ILE or COBOL ILE program.

The CA 2E support reduces the reliance on the Web Administration Interface Web Services wizard, with a programmatic invocation of the IBM shipped scripts which perform the pertinent Web Service administration tasks:

- Install a Web Service (i.e. automatically create and deploy).
- Uninstall a Web Service

Install and uninstall Web Service administration tasks are available as new 2E commands.

Additionally, a CA 2E user can model Web Services within a 2E model enabling 2E facilities such as impact analysis to be applied to web services.

Installation Requirements

To enable Web services provider and requestor support you must install the products that are listed in the following link for i6.1 and higher:

http://www-03.ibm.com/systems/i/software/iws/support.html

To see the list of installed products on your machine, you should run the following command:

```
GO LICPGM
```

Then select option 10.

RPG and COBOL source must be is compiled with the PTF's listed below to include the information necessary to generate Web services programs or service programs.

Required IBM PTFs

In order to correctly support CA 2E (particularly Web Service Support) you need to ensure that you have the latest IBM Cumulative Package and Latest HTTP Group PTFs installed.

i6.1

5761SS1 - SI34865

For more information on PTF's, see IBM's website.

PCML in Module

At i6.1 and higher, *after* the PTF's are installed, but *before* the module source is compiled by CA 2E, the following statements *must* exist in the module source. The PGMINFO statement does not have to be manually added at i6.1 and higher.

RPG

H PGMINFO(*PCML:*MODULE)

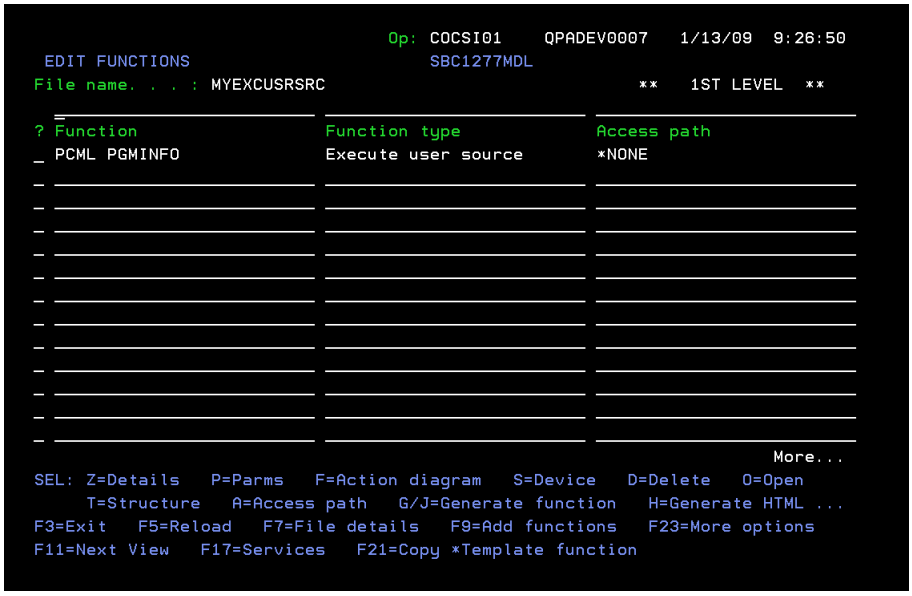
COBOL

PROCESS OPTIONS PGMINFO(PCML MODULE)"

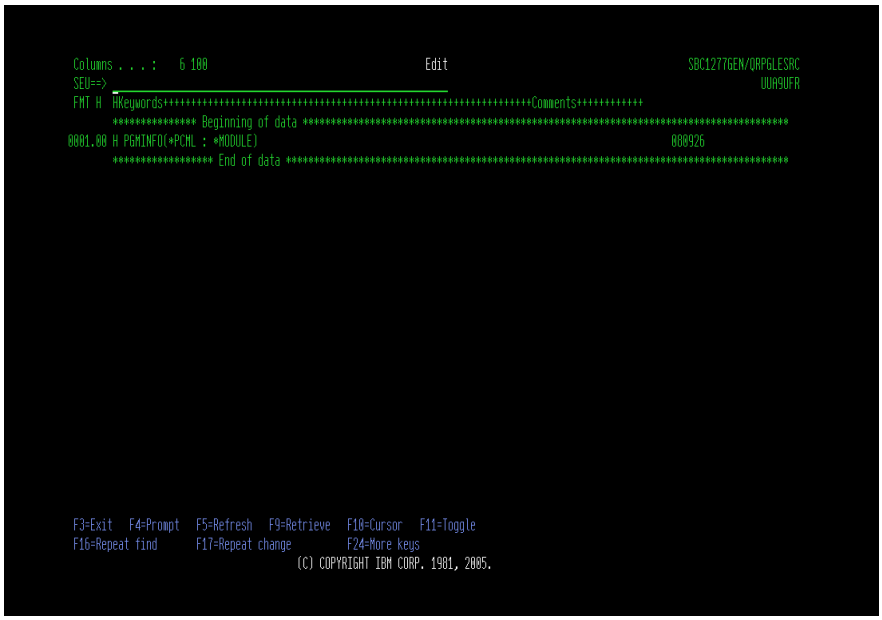
Note: 2E does not automatically generate these PGMINFO source lines.

An easy method to accomplish generation of PGMINFO into i6.1 and higher source is to create an EXCUSRSRC member containing the PGMINFO statement and then include that statement in the AD of the module, as in the following examples:

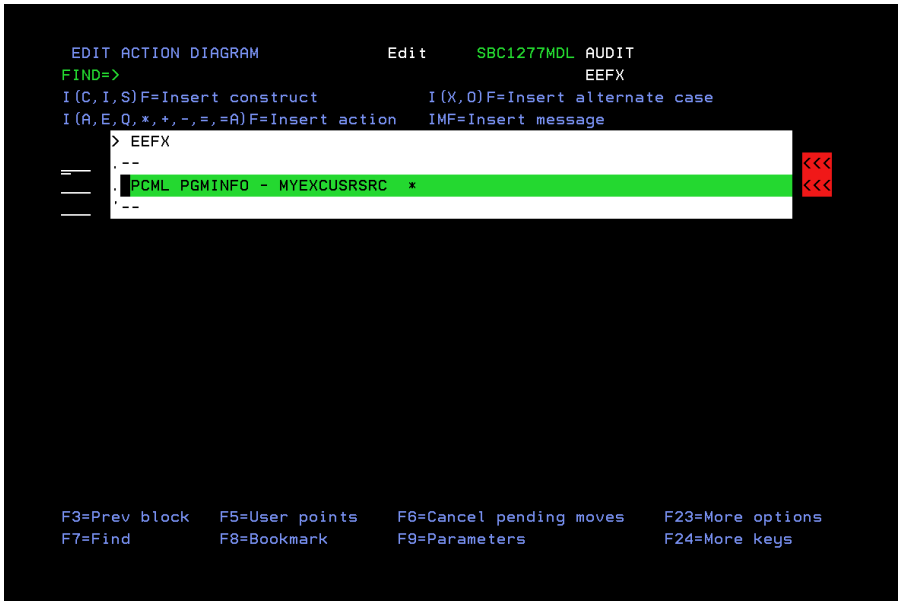
1. "PCML PGMINFO" is a function of type EXCUSRSRC.



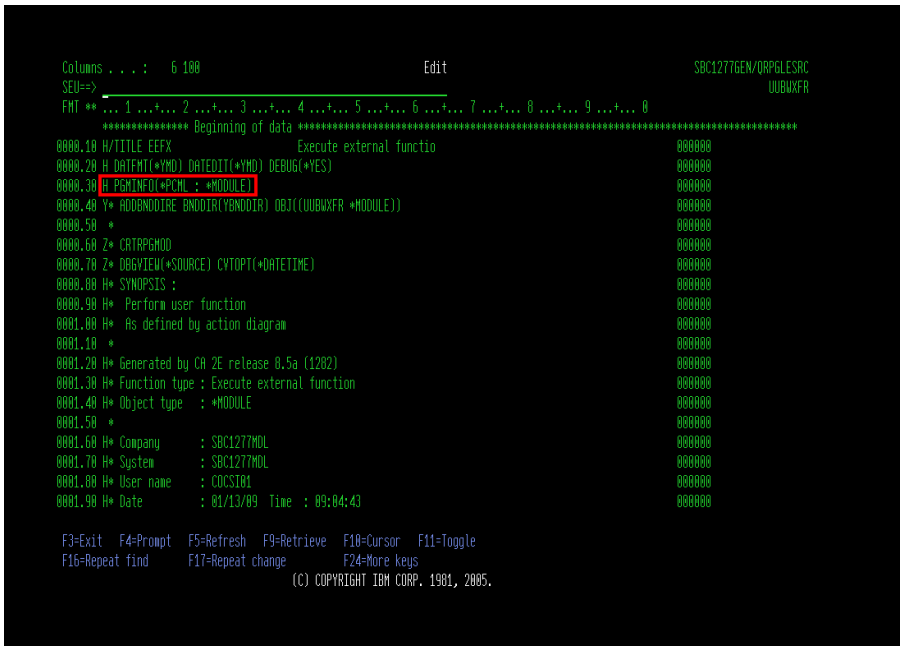
2. Source member for the EXCUSRSRC function contains "H PGMINFO(*PCML : *MODULE)".



3. AD of function EXCEXTFUN shows a call to the user source.



The generated source for the module shows the included PGMINFO line:



Note: At i6.1 and higher, it is not necessary to add the PGMINFO statement, but you must ensure that the Compiler Overrides for the Module have the PGMINFO parameter set to PGMINFO(*PCML *MODULE).

In addition to modifying CA 2E functions using EXCUSRSRC to include the keyword PGMINFO(*PCML : *MODULE), the YRP4HS2 model value can be used to alter H spec generation (for an RP4 module) on a model-wide basis.

YRP4HS2 (*MODULE) ships with a value of 'H DATFMT(*YMD) DATEDIT(*YMD) DEBUG(*YES)'.

Changing the value, as below, ensures any generated module is generated with contained PCML:

```
YCHGMDLVAL MDLVAL(YRP4HS2) VALUE('H DATFMT(*YMD) DATEDIT(*YMD) DEBUG(*YES)

PGMINFO(*PCML : *MODULE)')
```

Architecture

The fundamental components of this solution are the new 2E commands that encapsulate programmatic invocation of IBM's Web Service Administration scripts.

IBM ships the following scripts.

Script	Purpose
installWebService.sh	Create and deploy a Web Service
listWebServices.sh	List Web Services deployed to a Web Services Server
startWebService.sh	Start a deployed Web Service
stopWebService.sh	Stop a deployed Web Service
uninstallWebService.sh	Uninstall a deployed Web Service
createWebServicesServer.sh	Create Web Services Server*
deleteWebServicesServer.sh	Delete Web Services Server*
startWebServicesServer.sh	Start Web Services Server*
stopWebServicesServer.sh	Stop Web Services Server*

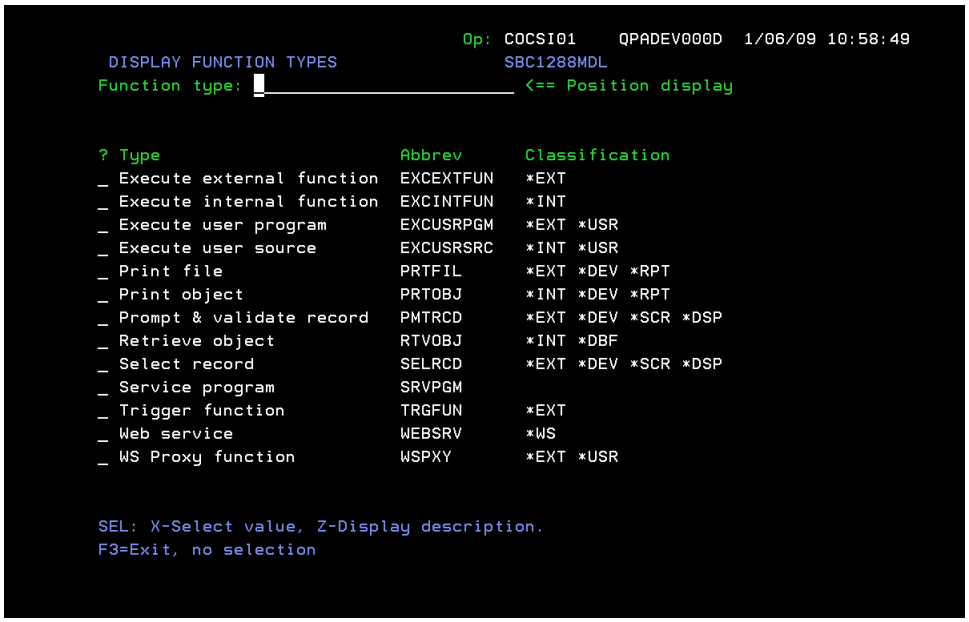
Note: It is *not* the intention of this phase of the CA 2E Web Service support to mimic all functionality available within the Web Administration interface. This release allows a Web Service to be modeled and to be installed to/uninstalled from a Web Services Server.

Two new 2E commands mirror and invoke functionality of two of the scripts:

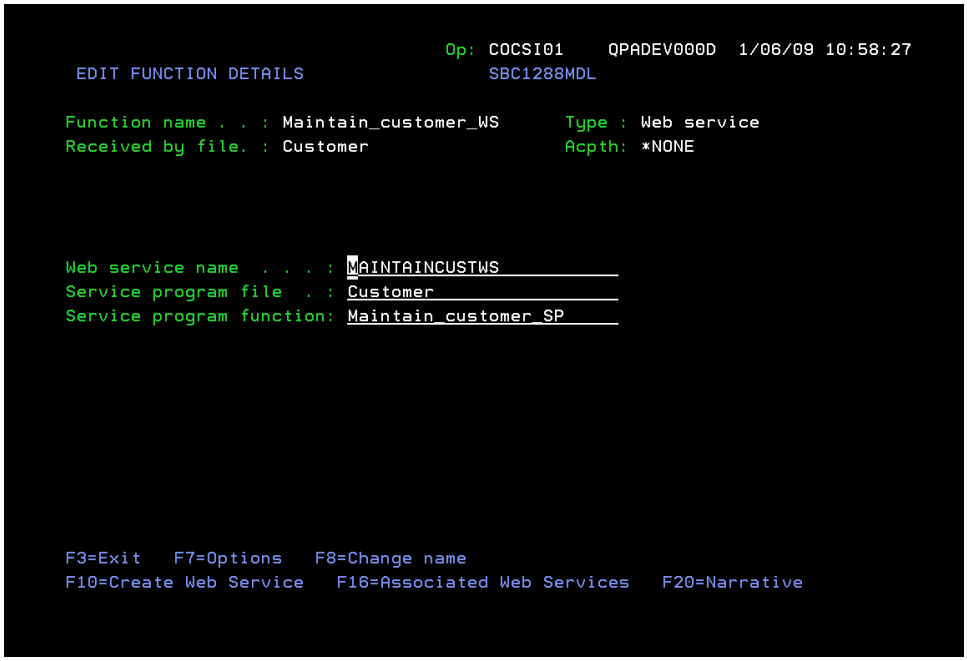
2E command	Purpose
YCRTWS	Create and deploy a Web Service (installWebService.sh)
YUNSWS	Uninstall a deployed Web Service (uninstallWebService.sh)

Note: The Web Service Administration interface (and therefore) scripts cannot operate on a remote environment; i.e. all WS work is for local machine.

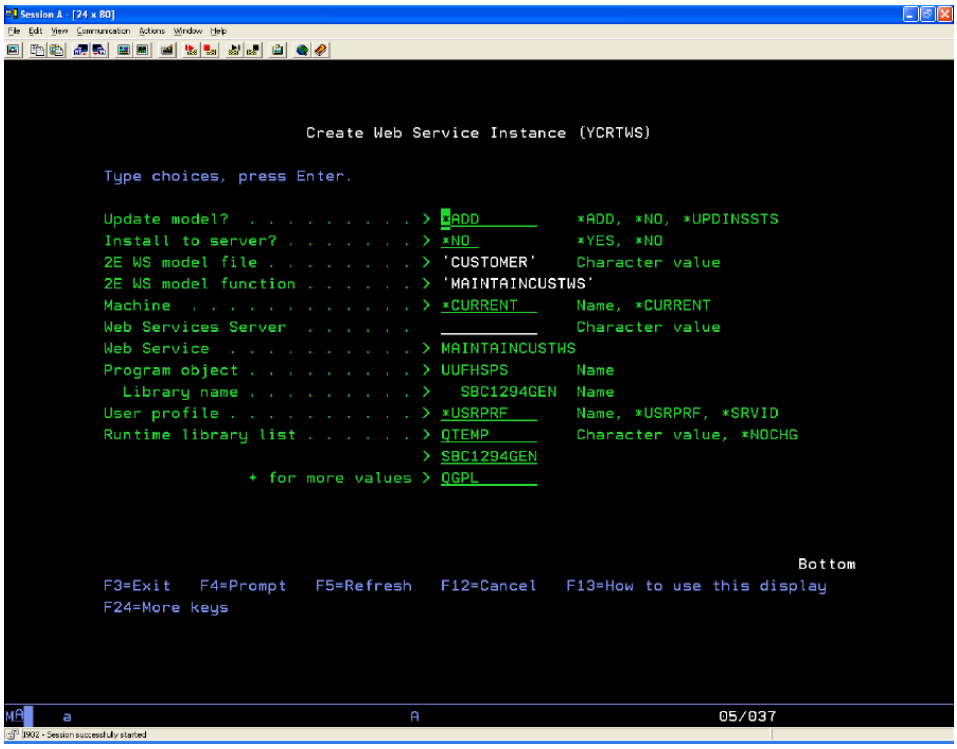
A new 2E function type of Web Service (WEBSRV) is available within the model.



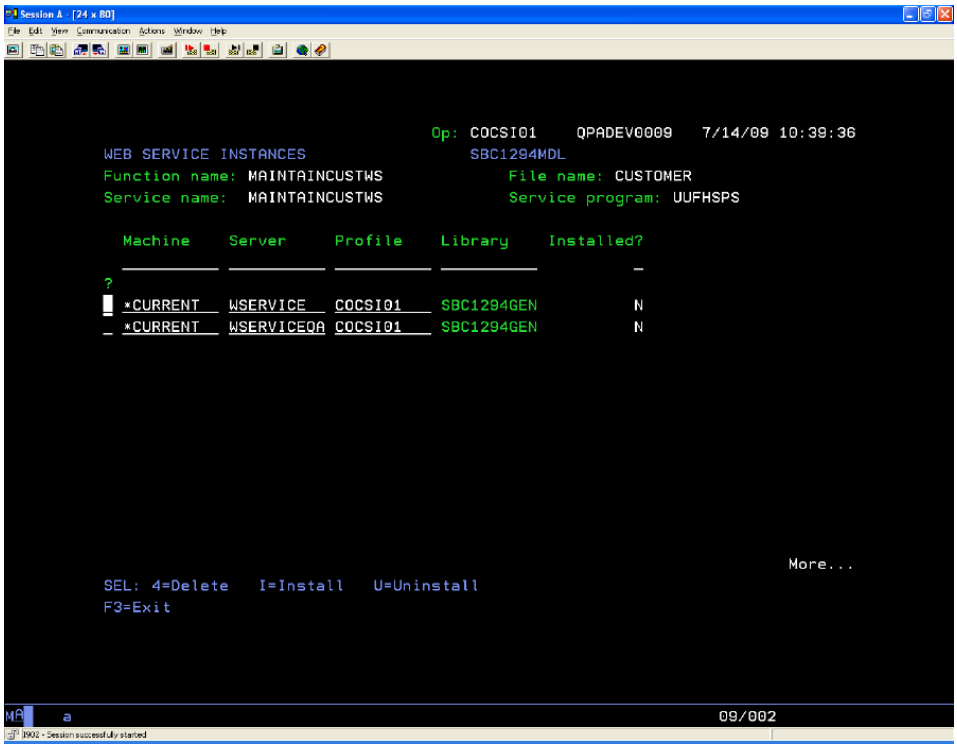
A function of type Web Service cannot be generated and does not have an action diagram. Its purpose is to serve as an umbrella for any number of Web Services Instances. A Web Service instance corresponds to a web service that is or can be deployed to a Web Services Server. The function of type web service has a customized EDIT FUNCTION DETAILS panel. This panel stores the default name of any created web service instance, as well as the 2E file/function that represents the target Service Program to be exposed.



The EDIT FUNCTION DETAILS panel provides option F10 to invoke YCRTWS.



The EDIT FUNCTION DETAILS panel also provides option F16 to invoke WEB SERVICE INSTANCES PANEL:



Web Services Limitations

This feature relies on the Web Services Server which is part of the IBM i operating system. Only 2E service programs generated as RPG ILE or COBOL ILE are candidates for exposure. The Web Service client portion is *not* created by this feature.

IBM states: "There are a few limitations within the Web services server regarding the deployment of services. To retrieve the most current information on restrictions, refer to the document located at */QIBM/ProdData/OS/WebServices/V1/server/docs/readme.txt*."

The YCRTWS and YUNSWS commands require the user issuing command to have special authorities *ALLOBJ and *IOSYSCFG. This is due to the underlying IBM IWS scripts requiring those special authorities.

One of these limitations you need to be aware of is this: When a module is defined with a RCD parameter definition, in the generated source a data structure is used as the parameter for the *ENTRY PLIST. If some of the fields on the RCD parameter data structure are not being used as parameters, then due to an IBM limitation in the PCML generation, in the IBM Test client interface you will see redundant parameters (or subfields) named "_unnamed_1", "_unnamed_2", etc. These can be ignored and do not cause any problems.

Full details of the restrictions can be viewed in IBM's documentation.

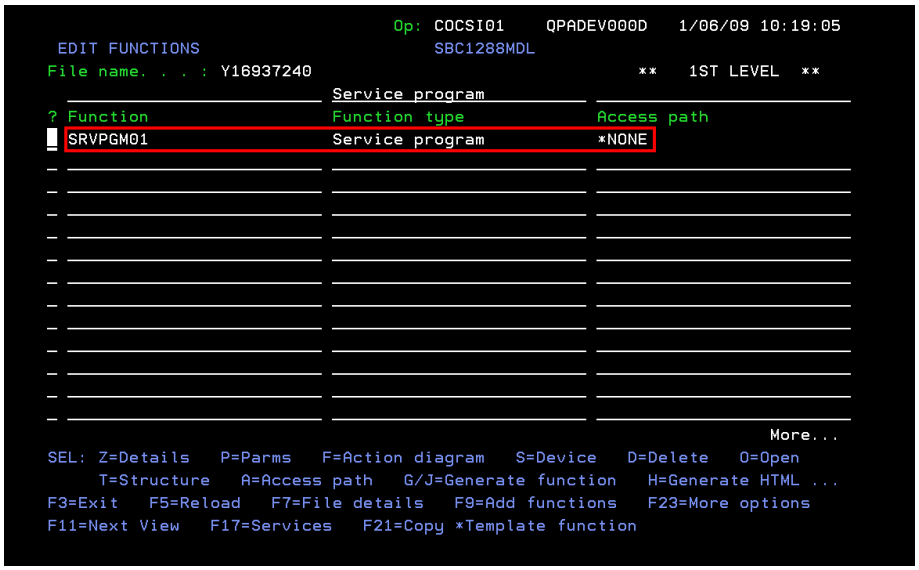
Note: To call a deployed web service on a web services server, the user ID that is on the Properties for the web service needs to satisfy the following criteria: 1. The user ID must have the necessary authority to this program object and any other additional program objects. 2. The server profile QWSERVICE must have *USE authority to the user ID.

Sample Flow

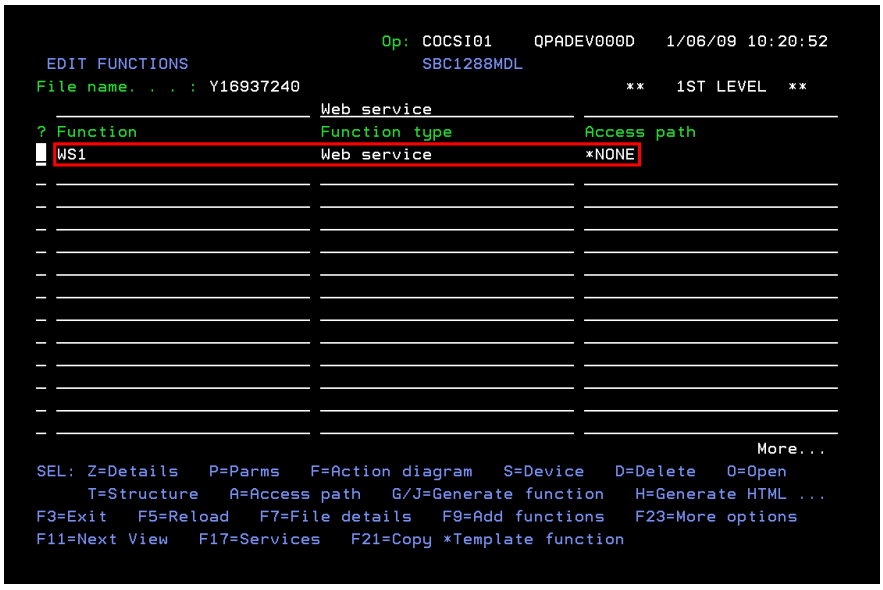
This section shows an example of how you might use these new features in CA 2E. Some settings and selection might vary depending on your system configuration.

To define and deploy a Web Service from CA 2E

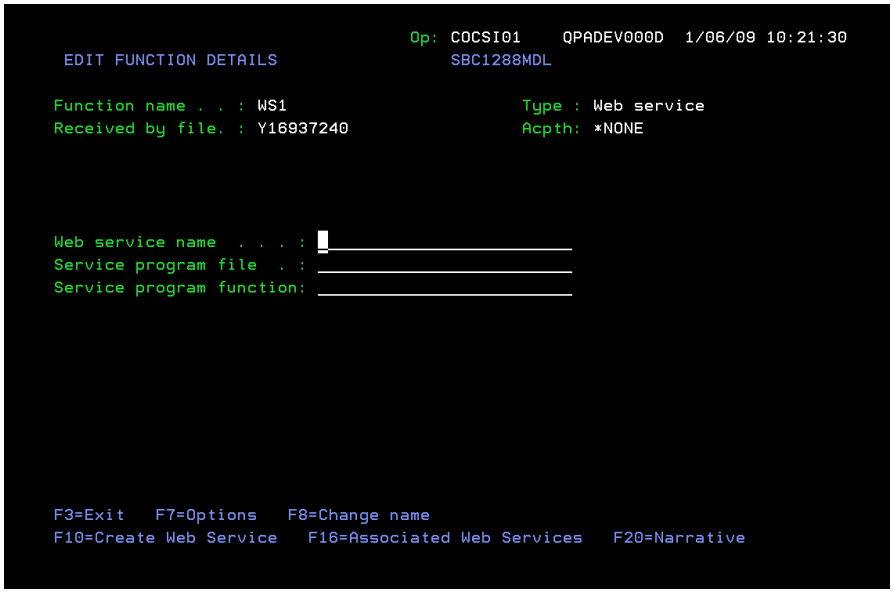
1. Identify a Service Program that has module operations you want to expose via a web service.



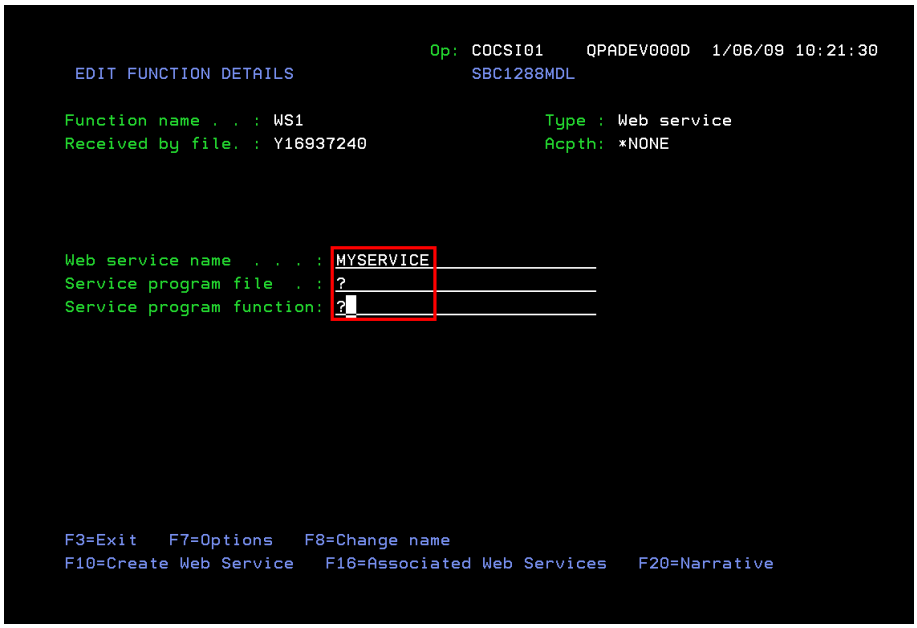
2. Create a new Web Service type function.



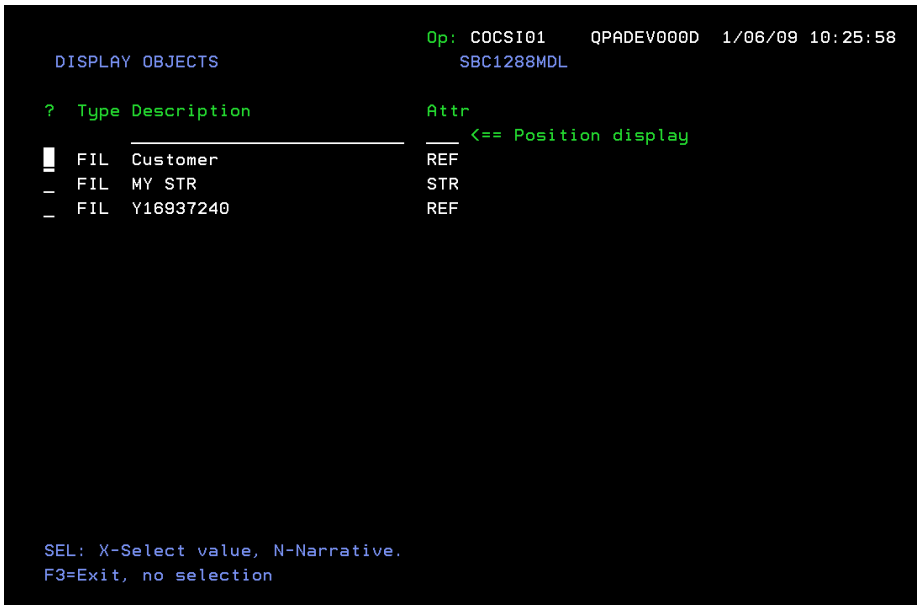
3. Zoom into the Web Service function to Edit Function Details.



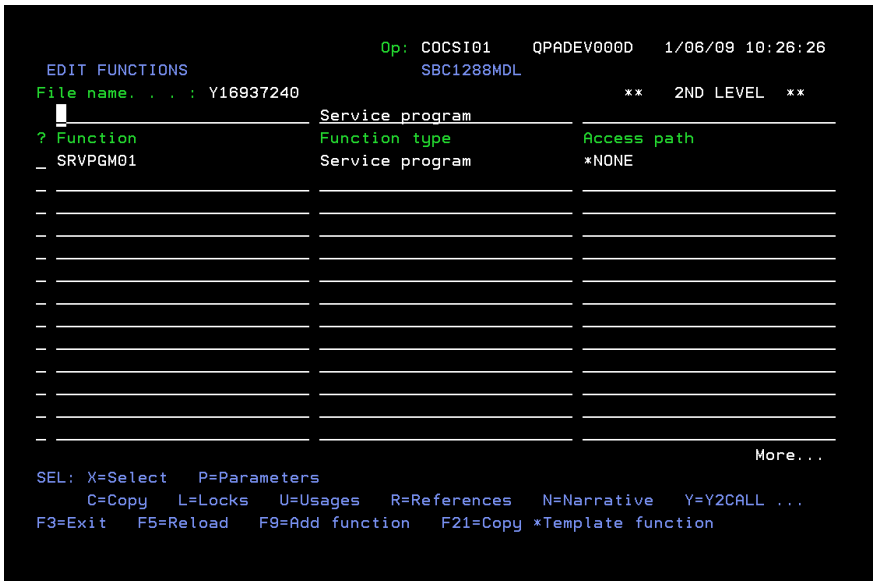
4. Specify the Web Service Name and Select Service Program.



Note: On Service program file and Service Program function allows user to choose file, as in the following example:

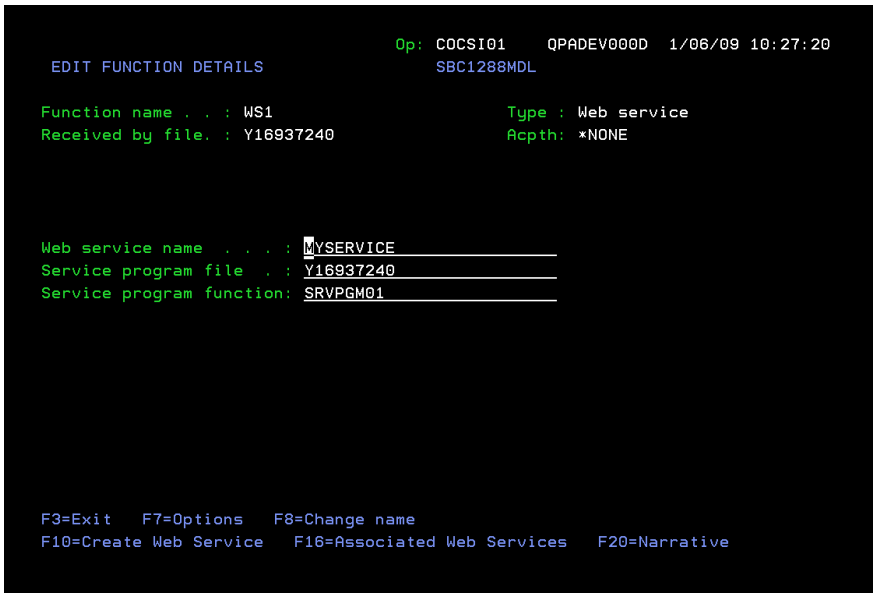


Users can also choose function, as in the following example:



5. EDIT FUNCTION DETAILS

   Auxiliary details populated.



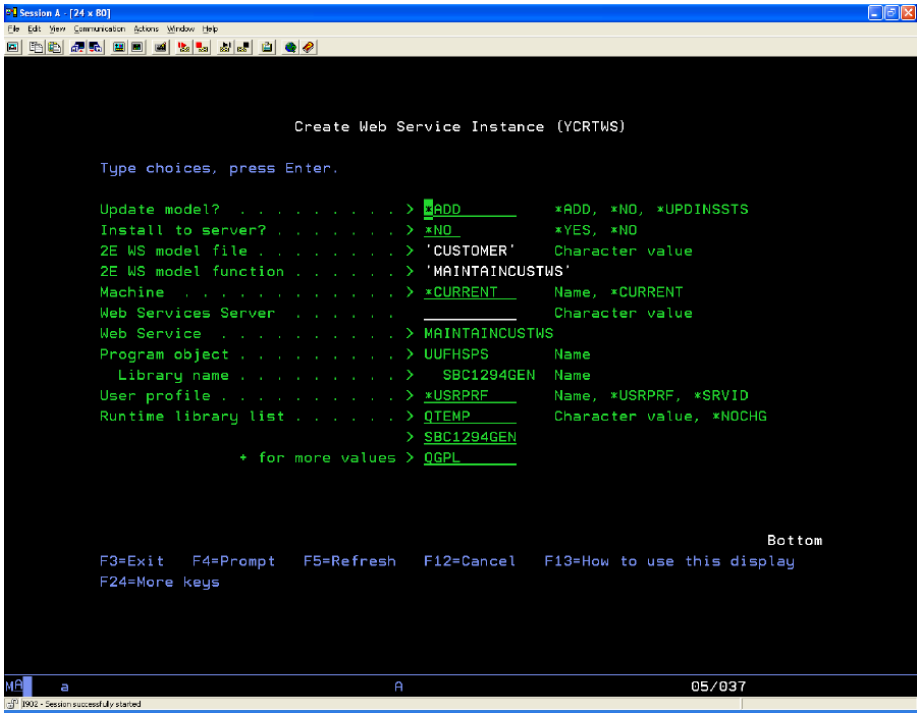
Note: Any number of Web Service Instances can be created from the EDIT FUNCTION DETAILS panel. Each Web Service will be associated with the 2E Web Service function in the panel header.

All the associated Web Services can be viewed using F16 to invoke the WEB SERVICE INSTANCES panel.

To create a Web Service Instance with a different Web Service name, you should change the Web Service Name on Edit Function Details, then press F10.

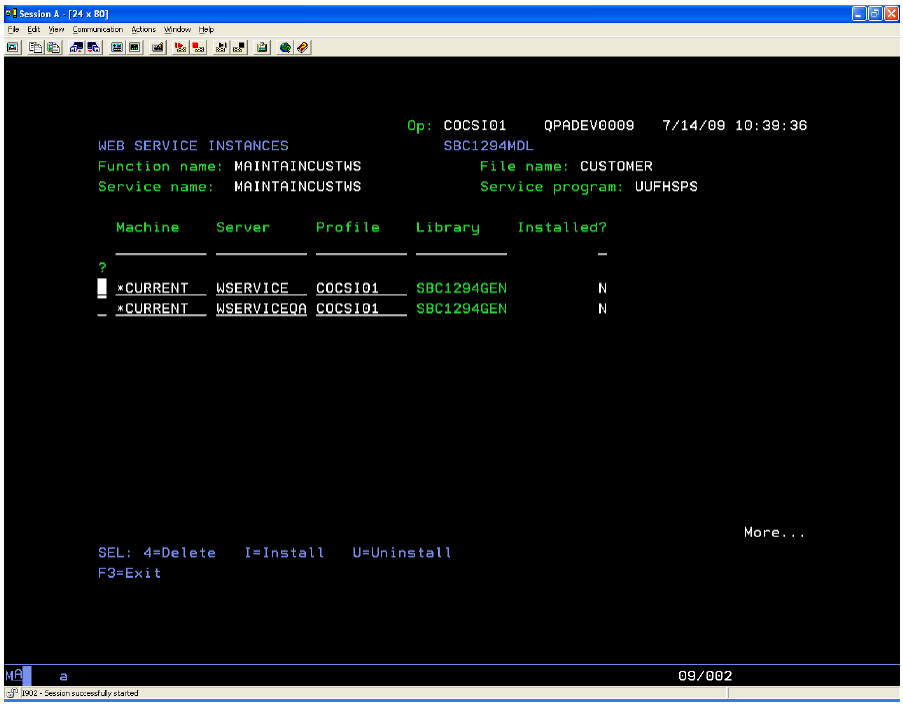
Note: If any of the instances have been installed to a server and the Installed Flag is set to Y, then the Web Service Name, Service Program File and Service Program Function all become Output only. This is to ensure that the Web Service details in the model, and the installed Web Service Instance, are in synch.

6. Create the Web Service Instance (YCRTWS)



Note: See the section *New Commands* for the command parameter descriptions.

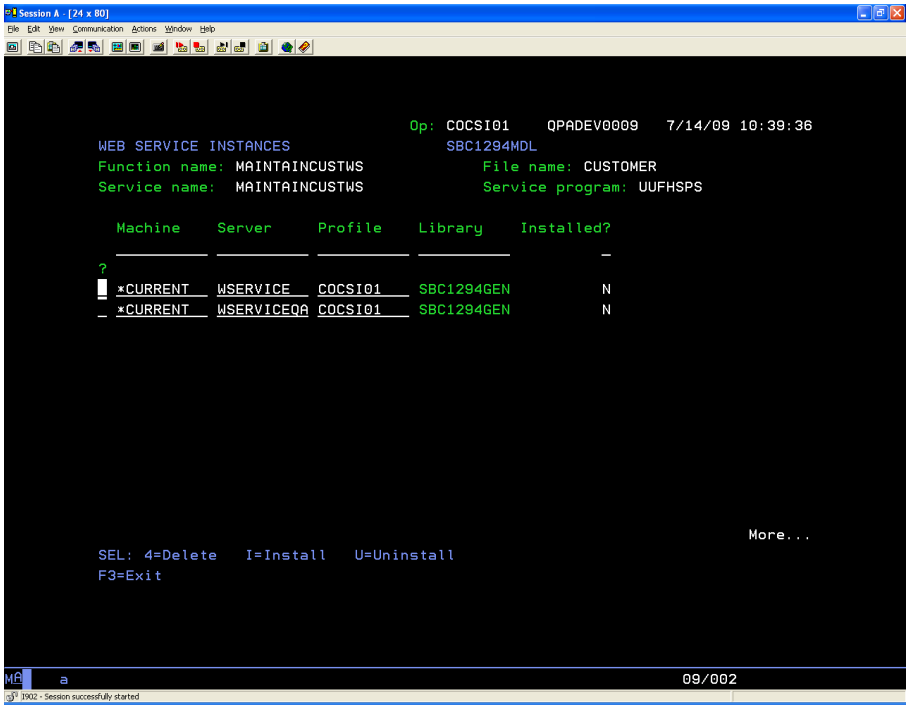
7. Work with Web Service Instances:



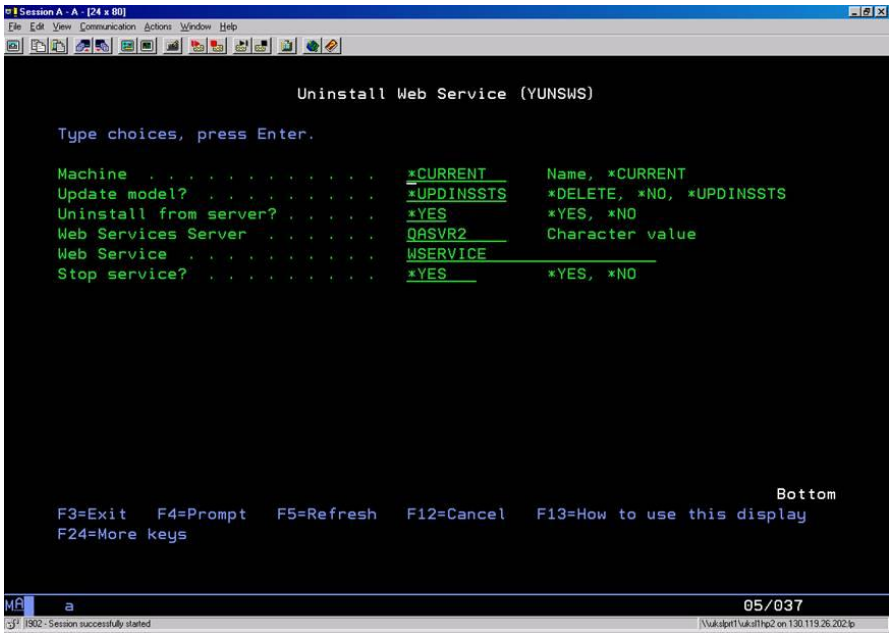
A machine name of *CURRENT indicates that the Web Service instance is to be deployed on the local machine. If you enter any other machine name other than the local one, the Installed flag will always be set to ?, since the machine is not available.

You can only take option I - Install for a Web Service Instance with Machine = *CURRENT. To install Web Service instances to a remote machine, you must use the Web Service Remote Deployment feature. For further details of this please refer to the corresponding section in this chapter.

8. Use option I to install a modeled Web Service to a Web Services Server.

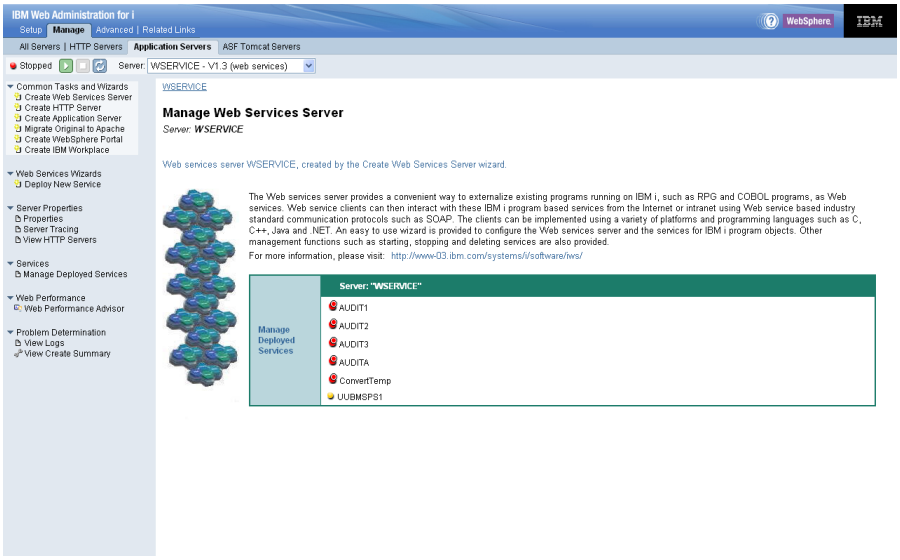


9. Use option U to uninstall a modeled WS from a Web Services Server.



10. Use IBM Web Administration interface to start/stop/test deployed Web Service.

    `http://{your_machine}:2001`



Commands

In addition to the install/uninstall functionality in the EDIT FUNCTION DETAILS and WEB SERVICE INSTANCES panels, the 2E Web Service commands can be invoked from the command line using the commands listed below:

Command	Function
YCRTWS	Installs a Web Service instance.
YUNSWS	Uninstalls a deployed Web Service instance.

YCRTWS (Create Web Service Instance)

The Create Web Service Instance (YCRTWS) command is used to install a web service to the IBM Web Services Server that contains an operation to invoke the RPG ILE or COBOL ILE program specified.

Update model? (UPDMDL)

Specifies if and how the model is updated.

***ADD**

New WS instance is added to the model (it must not already exist).

***NO**

The model is not updated at all.

***UPDINSSTS**

For a WS instance that already exists, the Installed status is updated.

Install to server? (INSTALL)

***YES**

The WS is installed to a Web Services Sever. The WS instance name must be unique to the specified Web Services Server.

***NO**

The web services server is not updated at all.

2E WS model file (MDLFIL)

Model-file-name

Specify the name of the 2E WS model file that owns the 2E WS model function.

2E WS model function (MDLFUN)

Model-function-name

Specify the name of the 2E WS model function to which the web service instance will be associated.

Machine (MACHINE)

Specifies the name of the machine onto which the web service instance will be deployed.

***CURRENT**

Refers to the local machine.

name

Specify the machine name. This can be the local machine or a remote machine. The machine name is not validated and the machine need not exist on the local, or indeed any network.

SERVER (char (10))

The name of the web services server in which the service will be installed.

server-name

Specify a web services server name.

SERVICE (char(25))

The name of Web service to be installed.

***PGMOBJ**

The program object name will be used.

service-name

Specify the name of the web service to be installed.

PGMOBJ (*PNAME)

The path to the ILE program or service program.

server-name

Specify the path to the ILE program or service program.

USRPRF (*VNM)

The user profile that the Web service will run under.

***USRPRF**

The web service will be created to run under the user profile that is invoking the YCRTWS command.

user-profile

Specify the user profile that the Web Service will run under.

Note: This user profile is granted access to all the Web service files and directories. If the service user ID is different from the server user ID, the server user ID must be given *USE authority to the service user ID.

***SRVID**

The Web services server user ID is used to run the service.

RTLIBL

A list of libraries, that will be added to the library list prior to invoking the Web service.

library-list

Specify the list of libraries.

***NOCHG**

No user-specified libraries will be added to the run-time library list.

YUNSWS (Uninstall Web Service)

The Uninstall Web Service (YUNSWS) command is used to uninstall a web service from the IBM Web Services Server that contains an operation to invoke the RPG ILE or COBOL ILE program specified.

The command can also update the installed status of the WS in the model or delete the modeled service entirely.

Update model?

This specifies if and how the model is updated.

***DELETE**

The WS instance is deleted from the model.

***NO**

The model is not updated at all.

***UPDINSSTS**

For a WS instance that already exists, the Installed status is updated

Uninstall from server? (UNINSTALL)

***YES**

The WS is uninstalled from the Web Services Sever.

***NO**

The web services server is not updated at all.

Web Services Server

The name of the web services server from which the service will be uninstalled.

server-name

Specify a web services server name

Web Service

The name of Web service to be uninstalled.

service-name

Specify the name of the web service to be installed.

Stop Service?

An indication whether the service should be stopped before an uninstall.

***YES**

The service is stopped before uninstall.

***NO**

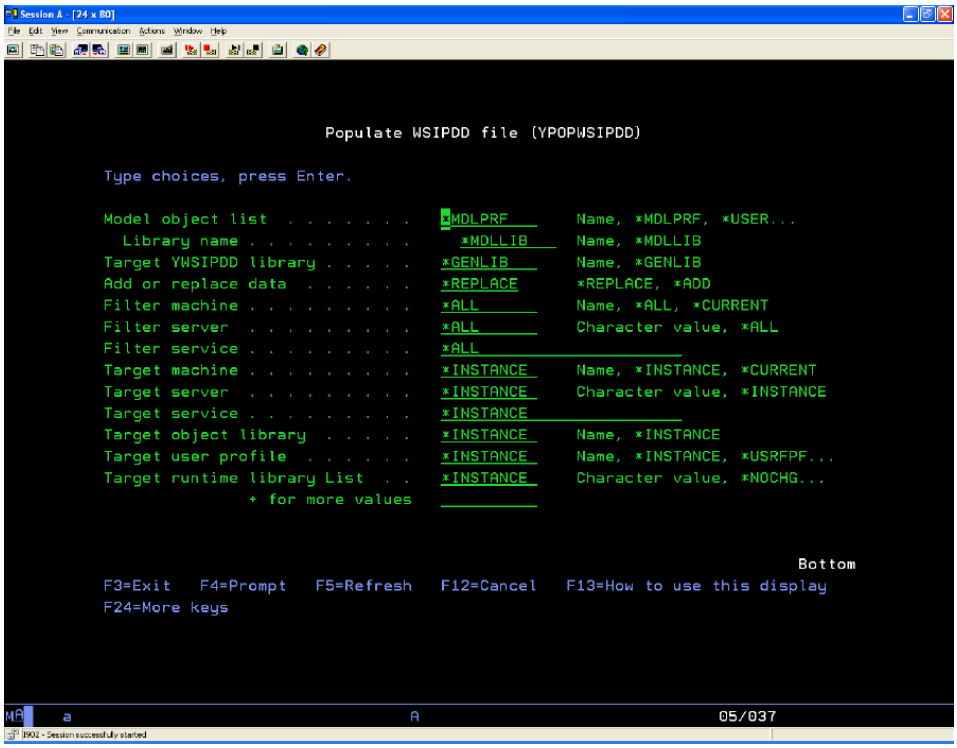
The service is not stopped before uninstall. An error will be returned if the service is active.

Running YUNSWS will delete or update the record in YWSICTLRFP file, providing UPDMDL is not *NO.

Web Service Remote Deployment

With Web Service Support in 2E, it is possible to bundle up Web Service instances that need to be ported and deployed to a remote machine. To do this you will need to use the Web Service Remote Deployment feature. Use the following commands to utilize this feature:

YPOPWSIPDD (Populate WSIPDD file)



Populates a Web Service Instance Portable Deployment Data file (WSIPDD). Once populated, a WSIPDD file can be moved to a remote machine, where the related YEXCWSIPDD (Execute WSIPDD) command can process the file to deploy web service instances on that remote machine.

Notes:

- Portable deployment does not require CA 2E or 1E to exist on the remote machine on which the YEXCWSIPDD command is running. However, the YEXCWSIPDD command does require certain application objects to exist on the machine on which the command is running. These objects can be created in a target library using the YDUPAPPOBJ command parameter, *WS argument. The YEXCWSIPDD command takes a WSIPDD file as an input.
- To run this command the YCA/CAWS/UserData and YCA/CAWS/ProdData/YQSHLOG folders need to exist in the IFS. Take extra care with this in the case where the command is being used on a remote machine that does not have 2E installed.

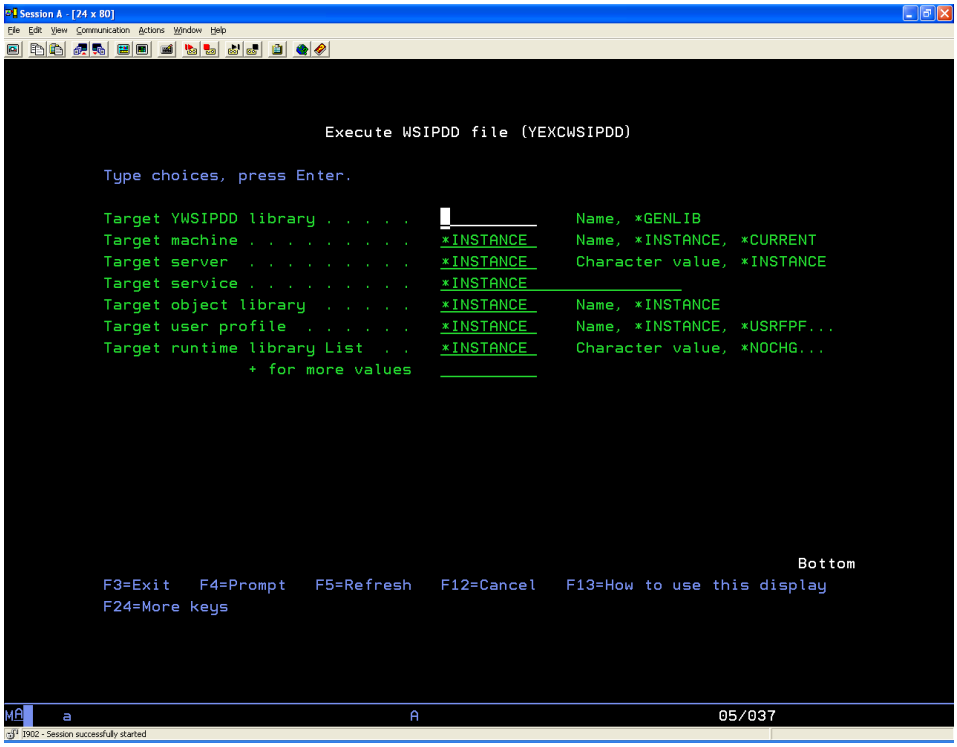
  For more information on how to restore the YCA structure, see the section "Web Services Support" in the *Installation Guide.*

The YPOPWSIDD command takes a model list as input. The model list is processed and for each function of type Web Service, additional processing occurs (items in the list that are not Web Service functions are ignored). For each Web Service function, all its web service instances are processed. Where a web service instance is not excluded due to filtering arguments on the YPOPWSIPDD command, an instance record will be created in the target WSIPDD file.

Note: The target WSIPDD file is always called YWSIPDDRFP, but the location is specified on the WSIPDDLIB parameter.

The Target parameters on the YPOPWSIPDD command allow the modelled web service instance data to be overridden when populated to the WSIPDD file.

YEXCWSIPDD (Execute WSIPDD file)



This command executes a Web Service Instance Portable Deployment Data file (WSIPDD), to deploy web service instances. The WSIPDD file should have been populated by the CA 2E YPOPWSIPDD command. Typically the WSIPDD file is then moved to a different machine to be executed by the YEXCWSIPDD command.

Note: Portable deployment does not require CA 2E or 1E to exist on the remote machine, on which the YEXCWSIPDD command is running. However, the YEXCWSIPDD command does require certain application objects to exist on the machine on which the command is running. These objects can be created in a target library using the YDUPAPPOBJ command (see the *WS argument for the DUPOPT) parameter. The YEXCWSIPDD command takes a WSIPDD file as an input.

Note: The input WSIPDD file is always called YWSIPDDRFP, but the location is specified on the WSIPDDLIB parameter.

For each record in the WSIPDD file with an ACTION flag of 'I' a web service instance will be deployed by the YCRTWS command. The Target parameters on the YEXCWSIPDD command allow the WSIPDD web service instance data to be overridden when deploying.

Note: If a web service instance is successfully deployed as a result of the YEXCWSIPDD command instance record's Action flag is updated to BLANK. The YINZWSIPDD command can be used to reset the WSIPDD Action flag.

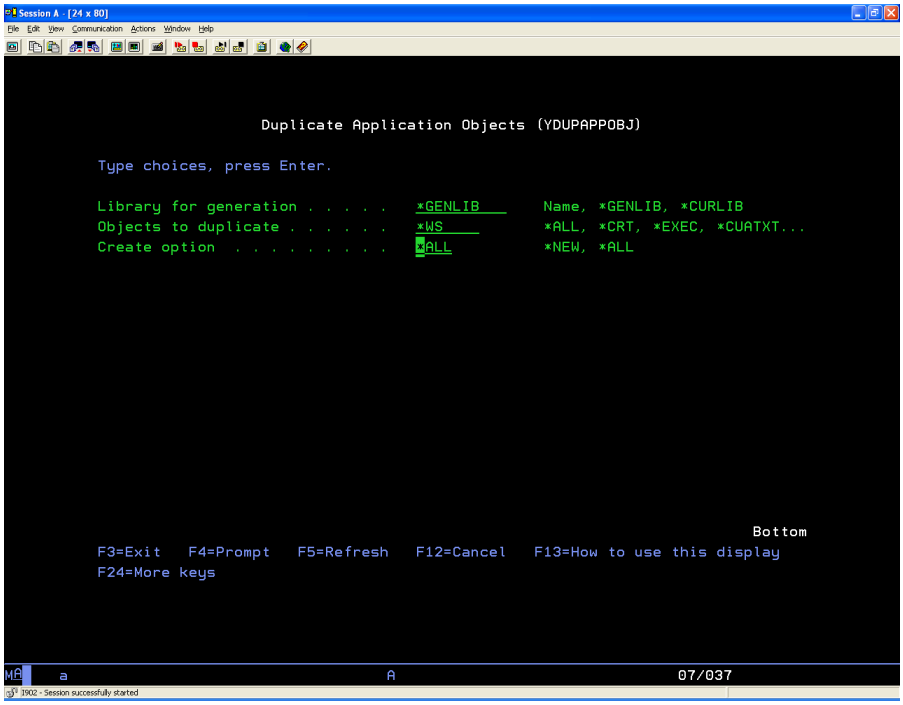
YINZWSIPDD (Initialise WSIPDD file)

The YINZWSIPDD command can be used to reset the Action flag on records in a WSIPDD file. See the YEXCWSIPDD command for more information.

To bundle up and deploy your remote Web Service instances

1. Create the Web Service Application objects using YDUPAPPOBJ

   There are a number of objects that need to be created in order to use Web Service Remote Deployment. These should be created by running the YDUPAPPOBJ command as follows:



You can either create these objects into the Model Generation library, or into a new library to be used for deployment.

2. Build a model list containing all the Web Service functions whose modeled Web Service instances you want to portably deploy.
3. Use the YPOPWSIPDD (Populate WSIPDD file) command to populate the target YWSPDD file.

   See the command description above for details.
4. Copy the library that contains the WSIPDD file (this will be the library that you created your WS application objects into) to the remote machine.
5. On the remote machine, use the YEXCWSIPDD (Execute WSIPDD file) to deploy each of the Web Service instances in the portable deployment file. See command description above for details.

The YEXCWSIPDD command calls the YCRTWS command for each record in the portable deployment file, which installs the WS Instance on that machine. If a Web Service already exists on the machine, the deployment for that Instance record will fail, and the Action field will be left as 'I'.

References

- IBM information Center
- Integrated Web Services for i
- Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", 26 June 2007, W3C
- Web Services Architecture, W3C Working Group Note", 11 February 2004, W3C

Standards and Web Services

Chapter 6: IBM i Database Trigger Support

In the i OS operating system, a trigger is a set of actions that execute automatically when a program performs a specified change operation on a specified database file. The change operation can be an *insert*, *update*, *delete* or *read* high level language (HLL) statement in an application program. You can design triggers to do almost anything—some uses for triggers include:

- Enforcing business rules
- Validating input data
- Writing to other files for audit trail purposes

Some benefits of using triggers are:

- Faster application development: Because triggers are stored in the database, actions performed by triggers do not have to be coded in each database application
- Global enforcement of business rules: A trigger can be defined once and then reused for any application using the database
- Easier maintenance: If a business policy changes, it is necessary to change only the corresponding trigger program instead of each application program
- Improve performance in client/server environment: All rules are run in the server before returning the result

This section contains the following topics:

Implementing Triggers

A trigger is attached to a file using the IBM Add Physical File Trigger (ADDPFTRG) command. This command specifies the database change operation that must occur for the trigger to fire. It also specifies the time the trigger should fire relative to the database change operation (before or after the database change has occurred). A trigger program defines the set of actions to perform when the trigger is fired. Trigger programs are named in the ADDPFTRG command.

When an application program makes a change to the data in a database file, i OS Data Management (DM) checks for the existence of a trigger for the file. If a trigger exists, DM then calls the specified trigger program. The application program never explicitly calls the trigger program.

IBM defines the parameters that must be passed by DM to all trigger programs as follows:

- Parameter 1: Trigger buffer
- Parameter 2: Trigger buffer length

The trigger buffer parameter consists of a fixed-length portion and a variable-length portion. The fixed-length portion contains various fields that describe the trigger. The fixed-length portion contains a series of offset/length pairs that define where the old record format (ORF) and the new record format (NRF) are stored within the variable-length portion.

The variable-length portion contains the ORF and NRF values themselves as well as null-byte maps for each record format. The trigger buffer length parameter defines the overall length of the trigger buffer parameter. With these parameters, the trigger program has access both to information about the trigger itself and also full details of the data being updated, deleted, inserted, or read.

In addition to performing additional processing to that defined in the application program that is changing the database, a trigger can actually cancel the database change and signal to the application program that the change was unsuccessful. Under some circumstances, the trigger can also change the data that is being written to the database, overriding the data used in the update statement in the application program.

Typical Trigger Implementation

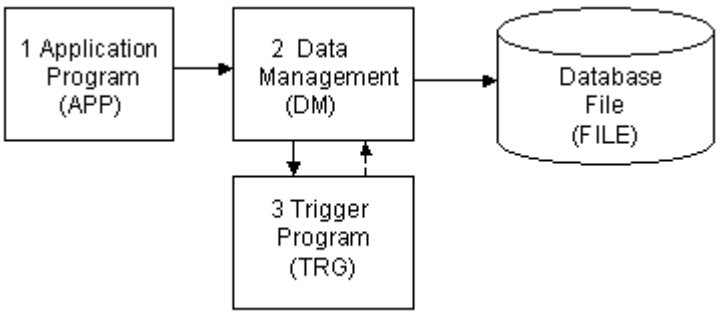
User-written trigger programs typically perform semi-generic processing to retrieve the file-specific ORF and NRF from the trigger buffer and move them into named structures prior to performing trigger-specific processing. This retrieval processing often involves relatively high complexity. Such processing can require the use of relatively esoteric functionality like pointer-based variables and dynamic memory operations not normally found in most HLL programming.

As a result of IBM's implementation of triggers (and particularly the way in which DM passes parameters to the trigger program), developers typically write separate trigger programs for each database file. A single trigger program may be used for more than one trigger on that file, however.

This implementation also makes it difficult to test trigger programs, since any test-harness must exactly duplicate the trigger buffer parameter specific to the file to which the trigger is attached.

In addition, the actual insertion of a trigger into the database (either initially or following a change in a trigger) requires multiple file locks on the database file and on all related access paths, often requiring the database to be taken offline during trigger implementation.

The following diagram shows the structure of a *typical* (non-CA 2E) trigger showing how the various components interact.



The processing flow is as follows:

1. Application program (APP) executes a database change statement (an INSERT, DELETE or UPDATE), resulting in a low-level call to i OS Data Management (DM).
2. DM calls the trigger program (TRG) specified for the trigger.
3. TRG performs user-defined processing and then returns control to DM.
4. If DM receives an error code from TRG (indicating that an error occurred during the processing in TRG), it does not update the file and instead sends the error code back to APP. Otherwise, it updates the file and returns control to APP.

CA 2E Trigger Implementation

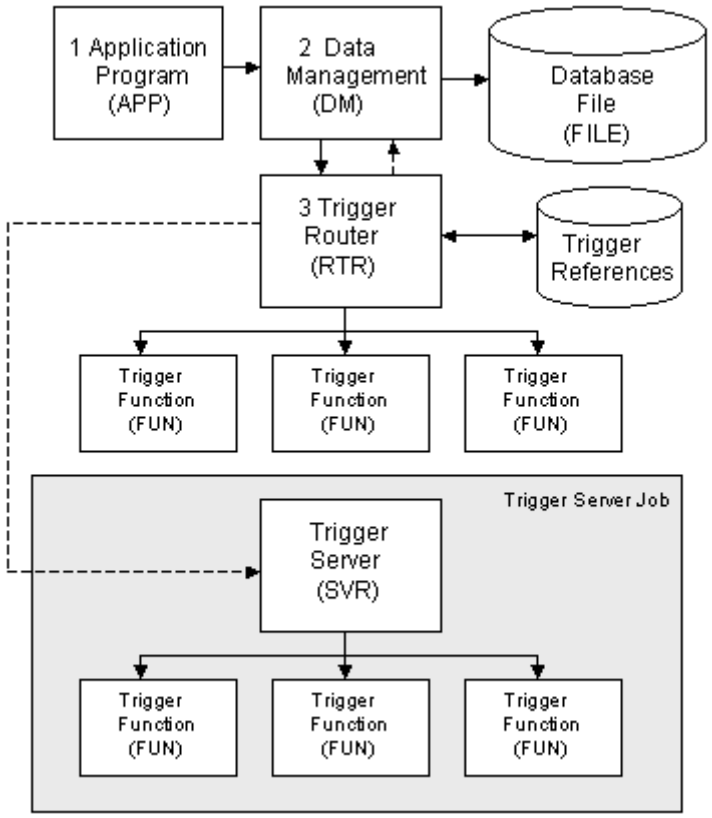
The CA 2E trigger implementation replaces multiple user-written, file-specific trigger programs with a single, generic file-independent Trigger Router that you can specify as the trigger program for any trigger over any database file. The Trigger Router uses a rule-based system held in a Trigger References File to call one or more further Trigger Functions that have previously been created in the CA 2E model. These functions are called either directly or asynchronously through a separate Trigger Server. You can specify each Trigger Function to perform any user-defined processing.

The parameters passed from the Trigger Router to a Trigger Function are simplified file-specific individual parameters. They represent the various elements of the trigger buffer parameter that was passed from DM to the Trigger Router. This simplification allows a test-harness to be generated in CA 2E very easily.

This implementation is also flexible enough to allow additional trigger functionality to be added, changed and tested very easily, simply by creating new Trigger Functions and updating the Trigger References File to link the Trigger Function to the trigger.

Note: If the Trigger Router routes a Trigger Function to the Trigger Server, the server cannot return an error code to the Trigger Router (and thence to the application program). Consequently, any processing within such Trigger Functions must be non-critical, since any failure within this processing will not be able to roll back the database change.

The following diagram shows the structure of a CA 2E trigger showing how the various components interact.


1 Application Program (APP)
2 Data Management (DM)
Database File (FILE)
3 Trigger Router (RTR)
Trigger References
Trigger Function (FUN)
Trigger Function (FUN)
Trigger Function (FUN)
Trigger Server Job
Trigger Server (SVR)
Trigger Function (FUN)
Trigger Function (FUN)
Trigger Function (FUN)

The processing sequence for a CA 2E trigger is as follows:

1. An application program (APP) executes an INSERT, UPDATE or DELETE statement, resulting in a low-level call to i OS Data Management (DM).
2. DM calls the Trigger Router (RTR) specified for the trigger.
3. RTR checks the Trigger References File to see if there are any Trigger Functions (FUN) to call for this trigger. There may be more than one FUN that should be called for a trigger. For each FUN record, if the FUN calling method is 'CALL', process steps 3.1. through 3.3. If the FUN calling method is 'DTAQ', process steps 3.4. through 3.5.
    a. RTR calls FUN directly, passing the pre-determined setoff parameters specific to the database file being processed
    b. FUN performs user-defined processing and returns control to RTR. If processing is unsuccessful, FUN returns error code to RTR
    c. If RTR receives an error code from FUN, it does not process any more FUN records, but returns the error code to DM. Otherwise, RTR returns to step 3 to process any subsequent FUN records in the Trigger References File
    d. RTR places an entry on the Trigger Data Queue. The data queue entry contains the same parameters as would be passed in a CALL to FUN.
    e. RTR returns to step 3 to process any subsequent FUN records in the Trigger References File.
4. If DM receives an error code from TRG, it does not update the file and instead sends the error code back to APP. Otherwise, it updates the file.

There are three separate sections to the CA 2E trigger support:

- CA 2E Model Support
- Model to Runtime Conversion
- Run Time

CA 2E Trigger Limitations

1. Triggers are *only* supported when the physical files on which trigger is implemented, are present in the generation library, denoted by model value (YGENLIB).
2. When we convert the database from DDS to DDL or SQL, the access path objects get recreated in SQL collection library, denoted by the model value YSQLLIB. However, because of a limitation in trigger processing algorithm, the trigger functionality can only be implemented on access paths present in generation library, denoted by the model value YGENLIB. Therefore, trigger implementation fails when you convert the database from DDS to DDL or SQL and the SQL collection library is different from the Generation Library.

The following lists the steps that you must perform for the trigger functionality to work after converting the database from DDS to DDL/SQL:

1. Ensure that the SQL collection library is same as the Generation library, (that is, both model values YSQLLIB and YGENLIB hold the same value) before regenerating your access paths as DDL or SQL. This ensures that the DDL/SQL type access paths objects are recreated in the same library where the original DDS-based access paths existed.
2. After converting the database to DDL or SQL, delete the trigger reference by using YWRKTRGREF command and choosing option 4=Delete against the access path that is being converted from DDS to DDL/SQL.
3. Rerun the YCVTTRGDTA command.

CA 2E Model Support

Model Support provides the ability, within the CA 2E model environment, to create and change Trigger Functions, to assign triggers to CA 2E database file definitions, and to manage Trigger Functions. This section contains information about:

- Administrative Tasks
- Creating Trigger Functions
- Editing Trigger Functions
- Using Trigger Commands

Performing Administrative Tasks

Initially, an administrator needs to run the YDUPAPPOBJ command specifying DUPOPT(*ALL) CRTOPT(*ALL). This copies the required application objects (including an empty copy of the Trigger References File YTRGCTLP) into any existing application libraries. Once you do this, no further administration tasks are required.

Creating Trigger Functions

You can create a Trigger Function by specifying either "Trigger Function" or "TRGFUN" for the Function Type, in exactly the same way that you would create any other CA 2E function. When you enter a Trigger Function using the Action Diagram Editor (ADE), you can include any processing that you would include in any other non-interactive function (including calling any other non-interactive function). You have access to all the *Trigger Control Data and ORF parameters as input-only, and the NRF and return code parameters as input/output.

When you create a Trigger Function (TRGFUN), the CA 2E model automatically creates a file-specific parameter list for the function. The Trigger Router passes the list to the function as follows:

1. Return code.
2. Trigger Control Data structure (from the *Trigger Control Data system file).
3. Old record format (ORF) structure (from the owning file).
4. New record format (NRF) structure (from the owning file).

The Trigger Control Data structure contains fields retrieved from the parameters passed by Data Management to the Trigger Router, as well as some derived fields, as follows:

Field	Description
*Trigger File	Database file being updated
*Trigger File Library	Library of database file being updated
*Trigger File Member	Member of database file being updated
*Trigger Event	Database change event which caused trigger to fire
*Trigger Time	Trigger time relative to database change
*Trigger Commit Level	Commitment control level of database file being updated
*Trigger Timestamp	Timestamp of trigger firing
*Trigger Record Length	Length of database file record format
*Trigger Job Name	Name of job which updated database file
*Trigger Job User	User of job which updated database file
*Trigger Job Number	Number of job which updated database file
*Trigger App Program	Program which updated database file
*Trigger App Library	Library of program which updated database file

Note: The *Trigger Job fields are included to allow asynchronous Trigger Function calls to determine the name of the job that actually changed the database file, rather than using the job fields from the Trigger Server. The *Trigger App Program and *Trigger App Library fields allow Trigger Functions to make processing decisions based on the application program that caused the file change. In r 8.1, the *Trigger App Library is not currently used and is passed as blank to the Trigger Function.

Editing Trigger Functions

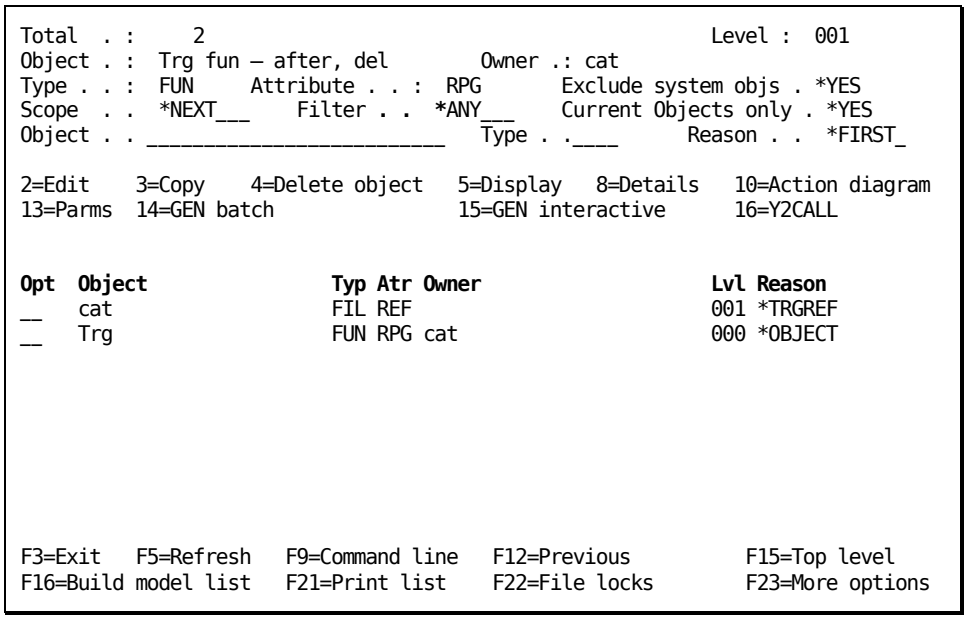
Once you create a Trigger Function, you can edit it using the Action Diagram Editor. From the EDIT FILE DETAILS panel, you can access a new EDIT FILE TRIGGER DETAILS panel to link Trigger Functions to a specific trigger for the owning file. This cross-reference link information is held in a CA 2E model file called YFILTRGRFP (MDL File Triggers) in an internal model-level format.

Press F18 from Edit File Details screen to get Edit File Trigger Details screen. Use this screen to link Trigger Functions to triggers based on the owning file.

```
EDIT FILE TRIGGER DETAILS

File name  . . . . . .: cat
Attribute  . . . . . .: REF
Source Library . . . . : UUA1SAMPLE
Distributed  . . . . . : N
Assimilated physical . . .:

D=Delete  F=Action Diagram
?        Time       Event     Cmt         Seq     Function
          A/B      D/I/R/U    Ctl
                              Y/N
_          A          D        N           1      Trg fun – after, del________
_          A          I        N           1      Trg fun 2 – after ins_______
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________
_          –          –        –           --      ____________________________

 F3=Exit   F4=Prompt   F5=Refresh
```

Within the Display Model Usages screen, any trigger references specified in the model (that is, any Trigger Functions are linked to a trigger on the file over which they were created from the EDIT FILE TRIGGER DETAILS screen) are displayed with reason *TRGREF.

```
Gen Objs :     1          Display Model Usages                Model:  <name>
```


Total . : 2 Level : 001
Object . : Trg fun – after, del Owner .: cat
Type . . . : FUN Attribute . . : RPG Exclude system objs . *YES
Scope . . *NEXT___ Filter . . *ANY___ Current Objects only . *YES
Object . . _____ Type . .____ Reason . . *FIRST_
2=Edit 3=Copy 4=Delete object 5=Display 8=Details 10=Action diagram
13=Parms 14=GEN batch 15=GEN interactive 16=Y2CALL
Opt Object Typ Atr Owner Lvl Reason
__ cat FIL REF 001 *TRGREF
__ Trg FUN RPG cat 000 *OBJECT
F3=Exit F5=Refresh F9=Command line F12=Previous F15=Top level
F16=Build model list F21=Print list F22=File locks F23=More options

Editing Trigger Parameters

When a new TRGFUN (Trigger Function) is created, the parameters are automatically derived from the based on file. If a database change occurs, however, it is necessary to visit the parameter details screen for the affected file. This automatically corrects the parameters for the function as shown in the following example. No other action is needed.

```
 EDIT FUNCTION PARAMETER DETAILS        <model name>

Function name. . : Trg fun – after, del    Type : Trigger function

Received by file . : cat                   Acpth: Physical file

Parameter (file) . : cat                   Passed as: RCD


   ?  Field                      Usage   Role   Flag error

      cat code                       I     MAP

      cat date                       I     MAP

      cat status                     I     MAP









SEL: Usage: I-Input, O-Output, B-Both, N-Neither, D-Drop.

  Role: R-Restrict, M-Map, V-Vary length, P-Position. Error: E-Flag Error.

F3=Exit

Parameter 'cat status' has been added for this trigger function.
```

Using Trigger Commands

The following commands affect Trigger Functions, Trigger Servers, and Trigger References:

Convert Trigger Data (YCVTTRGDTA)

The Convert Trigger Data (YCVTTRGDTA) command allows users to convert data from the CA 2E model internal file YFILTRGRFP into data in the run-time Trigger References File YTRGCTLP. The parameters for the YCVTTRGDTA command are:

Library for Data Model (MDLLIB)

This parameter is the name of a library containing the name of a design model from which the condition values are converted. The possible values are:

- *MDLLIB\-—Use the first model library found in the library list
- *CURLIB—Use the current library to invoke the job

Library for Generation (GENLIB)

Note: This library must contain the trigger runtime objects. These can be duplicated into the library by specifying the library as the target of the YDUPAPPOBJ command specifying DUPOPT(*TRG).

This parameter is the name of the library into which the command places converted values. The possible values are:

- *TRGLIB— Use the trigger runtime library named by the YTRGLIB model value in the model library.
- *GENLIB- Use the source generation library named by the YGENLIB model value in the model library.
- *CURLIB—Use the current library to invoke the job
- library-name - Specify the library into which converted values are placed.

Triggers to Convert (CVTOPT)

This parameter determines which trigger references should be converted from the model specified in the MDLLIB parameter into run-time trigger reference data in the YTRGCTLP trigger reference file in the library specified in the GENLIB parameter.

- *NEW—Only converts trigger references that do not currently exist in the run-time YTRGCTLP trigger reference file
- *ALL—Converts all trigger references in the model specified in the MDLLIB parameter into run-time trigger reference data. If any of the trigger references already exist in the YTRGCTLP trigger reference file, they are overwritten.
- *MDLLST—Checks the model list specified in the MDLLST parameter and only converts trigger references for Trigger Functions specified in the model list that were explicitly selected. If any of the trigger references already exist in the YTRGCTLP trigger reference file, they are overwritten.

Model Object List (MDLLST)

Note: This parameter is ignored unless CVTOPT(*MDLLST) is specified.

This parameter is the qualified name of the model object list to use. Trigger references are converted for any Trigger Functions existing in the list and that were explicitly selected. Any other object types in the list, or any Trigger Functions that were not explicitly selected, are ignored.

Possible model object list name values are:

- *MDLPRF—Special value meaning that the model object list name is retrieved from the user profile extension record for the current user is used as the model object list name.
- *USER—Special value meaning that the user profile name of the current user is used as the model object list name.

model-object-list-name

The name of the model object list to use.

Possible library values are:

- *MDLLIB—Special value meaning that the first model library in the current library list is used as the library for the object list
- library-name—Name of the model library that contains the model object list

Start Trigger Server (YSTRTTRGSVR)

The Start Trigger Server (YSTRTTRGSVR) command allows you to start one or more Trigger Server jobs. Once started, these jobs monitor the YTRIGGERQ data queue in the library specified in the command. The YSTRTRGSVR command parameters are as follows:

Trigger Data Queue Library (TRGLIB)

Specifies the name of the library containing the CA 2E Trigger Data Queue, YTRIGGERQ. If a Trigger Data Queue does not exist in the specified library, one is created.

Job Description (JOBD)

This parameter specifies the job description to use for the CA 2E Trigger Server job(s).

Since a Trigger Server runs as a continuous batch process until it is ended with the End Trigger Server (YENDTGRSVR) command, the processor overrides the job description you choose to use JOBQ(QSYS/QSYSNOMAX), to ensure that the Trigger Server is submitted using a job queue that allows multiple active jobs. All other job definition attributes are taken from the job description specified in this parameter.

The possible values are:

***USRPRF**

The job description for the user profile used by the job that is currently running is used for the trigger server job.

job-description-name

Specify the name (library-name/job-description-name) of the job description used for the trigger server job.

Number of Servers (NBRSVR)

Specifies the number of Trigger Server jobs that should be started by this command. All Trigger Server jobs use the same job description (as specified in the JOBD parameter) and will all monitor the same Trigger Data Queue YTRIGGERQ in the library specified in the TRGLIB parameter.

Since the Trigger Data Queue is a FIFO (first-in, first-out) data queue, if multiple Trigger Server jobs are running concurrently, each trigger request will be selected from the Trigger Data Queue by the first available Trigger Server job. There is thus no guarantee of the order in which the trigger requests will be processed, since this depends on many factors affecting the speed at which each Trigger Server job runs.

Consequently, if trigger requests must be processed in the same order in which the original trigger fired, you should process the trigger requests synchronously as a direct call by the Trigger Router, or you should ensure that only a single Trigger Server job monitors a specified Trigger Data Queue.

Running more than one Trigger Server job concurrently can improve system performance where many asynchronous trigger requests can appear at once. However, it will not affect the performance of the job in which the trigger was fired.

The values are:

***DFT**

A single Trigger Server job is started to monitor the Trigger Data Queue in the library specified in the TRGLIB parameter.

***MAX**

9 Trigger Server jobs are started to monitor the Trigger Data Queue in the library specified in the TRGLIB parameter. The maximum number of Trigger Server jobs that can be started is 99, but we retained the value for this parameter as 9 to preserve existing functionality.

Number-of-trigger-server-jobs

Between 1 and 99 Trigger Server jobs can be started to monitor the Trigger Data Queue in the library specified in the TRGLIB parameter.

Clear Data (CLEAR)

Specifies whether data should be cleared from the Trigger Data Queue prior to starting the Trigger Server job(s).

The possible values are:

***YES**

Any data queue entries on the YTRIGGERQ Trigger Data Queue is removed before the specified number of Trigger Server jobs are started

***NO**

Any data queue entries on the YTRIGGERQ Trigger Data Queue are not removed before the specified number of Trigger Server jobs are started. Consequently, they are processed immediately when the Trigger Server jobs start

End Trigger Server (YENDTRGSVR)

The End Trigger Server (YENDTRGSVR) command allows users to end one or more previously started Trigger Server jobs. The parameters to the YENDTRGSVR command are as follows:

Trigger data queue library (TRGLIB)—Specifies the name of the library containing the CA 2E Trigger Data Queue YTRIGGERQ

Work with Trigger References (YWRKTRGREF)

The Work with Trigger References (YWRKTRGREF) command allows users to display, add, delete or change Trigger Reference data. This is the data held in the Trigger References File that links specific database triggers to one or more Trigger Functions. Data is initially placed into this file because the YCVTTRGDTA command was run.

The YWRKTRGREF command parameters are as follows:

Trigger File (TRGFIL)

Specifies the name of the physical file you want to edit CA 2E trigger references. The possible values are as follows:

- *ALL—Display all CA 2E trigger references
- *trigger-file-name* - Display the CA 2E trigger references for the specified file only

Reload Trigger References (YRLDTRGREF)

The Reload Trigger References (YRLDTRGREF) command forces the Trigger Router (YTRIGGER) to reload its internal memory with the latest data from the Trigger References File (YTRGCTLP).

Trigger Reference Data in the Trigger Router

When the Trigger Router is first invoked within a job (because a trigger fires and the Trigger Router is defined as the trigger program), it loads the data from the Trigger References File into internal memory. On subsequent invocations within the same job, it uses the data it stored in memory rather than re-accessing the Trigger References File.

This processing ensures the best possible performance, since file I/O to the Trigger References File is performed only once during a job, rather than every time the Trigger Router is invoked. However, if changes are made to the data in the Trigger References File, these changes will not be reflected in the data used by the Trigger Router.

If you have made changes to the data in the Trigger References File you can execute this command to ensure that the next time the Trigger Router is invoked, it will use the changed data.

Note: Since each job has its own instance of the Trigger Router (with its own internal memory), you must run this command within the job that caused the Trigger Router to be invoked.

Model to Run-Time Conversion

This process converts a CA 2E model trigger definition into a trigger reference held in the Trigger References File. The Trigger Router interrogates this file when a trigger fires to determine which Trigger Functions to call.

You can convert model data in the MDL File Triggers file YFILTRGRFP into run-time data in the Trigger References File YTRGCTLP. This conversion process involves expanding model references into CPF (i OS) object names. The conversion process can also include the actual creation of the triggers over the database files (specifying the Trigger Router as the trigger program in each case).

A copy of the Trigger References File is shipped in the CA 2E base product library and can be copied into each application library using the CA 2E YDUPAPPOBJ command.

Run-Time Support

This section covers all of the run time aspects of trigger support within CA 2E-generated application programs, including the implementation of the Trigger Router, Trigger Server and Trigger References File.

There are two elements to the CA 2E trigger run-time support:

- Trigger Router
- Trigger Server

Trigger Router

When an application program updates a database file and the file has a trigger attached to it that specifies the Trigger Router as the trigger program, the Trigger Router checks for any records in the Trigger References File for the trigger. For each record found, it takes the appropriate action by directly calling the specified Trigger Function or sending a request for the Trigger Function to be called by the Trigger Server, by passing the Trigger Function parameters as an entry in the Trigger Data Queue.

If the Trigger Router is called but no matching record exists for the database file in the Trigger Reference File, the Trigger Router checks the value of the YTRGERR data area. If the YTRGERR data area has a value of *WARN, a message is sent to the job log stating that the Trigger Reference File data is missing. If the YTRGERR data area has a value of *ERROR, the Trigger Router will throw an error and the database file transaction will not continue.

The Trigger Router is implemented specifying ACTGRP(*CALLER) and USRPRF(*OWNER), according to IBM recommendations. All *PUBLIC access to the Trigger Router is *EXCLUDE.

Trigger Server

When the Trigger Server starts, it looks for entries to appear on the Trigger Data Queue. When an entry appears (placed there by the Trigger Router), the Trigger Server calls the specified Trigger Function and then returns to monitor mode.

```
DISPLAY CONVERT MODEL DATA MENU

              1. Convert model messages to database file.

              2. Convert condition values to database file.

              3. Convert distributed files to database file.

              4. Convert trigger data to database file.

Option:  _

F3=Exit    F6=Messages    F8=Submitted jobs    F9=Command line
```

Trigger Runtime Externalization

The 2E trigger support allows you to define a separate trigger runtime library, using the YDUPAPPOBJ command. This simplifies the copying of all trigger-related objects from your development machine to a production machine. The trigger runtime library contains all the objects required for trigger support.

The model value YTRGLIB (Trigger runtime library) is used to specify the name of the library in which the model trigger references are copied using the YCVTTRGDTA command. Before running the YCVTTRGDTA command, you must copy the trigger runtime objects into the YTRGLIB library using the YDUPAPPOBJ command, specifying DUPOPT(*TRG). You can assign a special value of *GENLIB to YTRGLIB model value to specify that the model source generation library will be used. By default, YTRGLIB is shipped with a value of *GENLIB.

Note: Multiple models can use the same trigger runtime library.

Chapter 7: Modifying Function Options

This chapter identifies the specific features of the standard function options that allow you to customize the functions in your model. This chapter also instructs you on how to specify these options.

This section contains the following topics:

Understanding Function Options

When a new function is defined, default options are set according to the function type and the model values. However, if your application requires it, you can change the default. You use the Change Model Value **(**YCHGMDLVAL**)** command to set the default value for certain options.

For more information on function types and the function options that apply to each type, see the chapter "Defining Functions."

Specifying Function Options

Function options are specified using the Edit Function Options panel. The options available depend on the function type.

Note: Some function options cause a corresponding section of the function's action diagram to be omitted or included.

For more information on action diagrams, see the chapter, "Modifying Action Diagrams."

Choosing Your Options

Use the following instructions to specify your function option choices.

1. Zoom into the file. At the Edit Database Relations panel, type **F** next to the selected file and press Enter. The Edit Functions panel appears.
2. Zoom into the function. Type **Z** next to the selected function and press Enter. The Edit Function Details panel appears.

   Note: You can also display this panel by entering option 2 for the selected function on the Edit Model Object List panel.
3. Press F7 to select options. The Edit Function Options panel appears.

   You can press F10 to toggle between a display of options available for the selected function and all available options. The current value of each function option is shown highlighted.

   Press F5 to view a list of the available standard header/footer functions. You use this display to explicitly assign a standard header/footer function to your function.
4. Select your options. Make your function option selections and press Enter. The Edit Function Details panel reappears.

Identifying Standard Function Options

The standard function options control the features of the standard functions. The following pages describe the standard function options and their available features.

Database Changes

The database changes function options determine whether the program provides add, change, and delete capabilities. You can select a combination of these features.

For edit type functions, all three features default to Yes. This means that, by default, all edit type functions allow add, delete, and change capabilities.

Create

This option specifies whether the function allows you to add new records to the database files on which the function is built.

- If Y is specified, the user can add database records with the function
- If N is specified, the user cannot add database records with the function

Change

This option specifies whether the function allows you to change existing records on the database files on which the function is built.

- If Y is specified, you can change database records
- If N is specified, you cannot change database records

Delete

This option specifies whether the function allows you to delete existing records from the database files on which the function is built.

- If Y is specified, you can delete database records with the function
- If N is specified, you cannot delete database records with the function

Display Features

The display features function options determine whether a function should include such features as a subfile selector column or a confirm prompt after data entry.

Confirm

This option specifies whether the function prompts for confirmation before updating the database files. A confirmation prompt appears at the bottom of the panel on which you can specify yes or no.

- If Y is specified, the function prompts the user for confirmation before updating the database files
- If N is specified, the function updates the database files without prompting for user confirmation

Initial Confirm Value

This option specifies the initial value that the confirmation prompt shows. This may be Y or N. The end user only needs to press Enter to accept the default value for newly created functions as specified by the YCNFVAL model value. This option only applies if the Confirm option is set to YES.

- If Y is specified, the initial confirmation prompt value is Y
- If N is specified, the initial confirmation prompt value is N
- If M is specified, the YCNFVAL model value is used

Note: If you have a National Language version of the product, the initial values reflect the national language version.

Standard Header/Footer Selection

Press F5 at the Function Options panel to select a non-standard header/footer for your function's device layout.

If Action Bar, What Type?

If the selected header/footer has an action bar, this option specifies the type of action bar to display. This option is available only for NPT generation.

- If A is specified, display a CA 2E action bar.
- If D is specified, display a DDS menu bar.
- If M is specified, the YABRNPT model value determines the type of action bar to display. Valid model values for YABRNPT are A or D.

Subfile Select

This option specifies, for functions that have subfiles, whether the subfile is to have a selection column on the left side of the function.

- If Y is specified, the subfile has a selection column
- If N is specified, the subfile does not have a selection column

Note: If Yes is specified for the Delete option, a subfile selection column must be specified.

Subfile End Implementation

This option specifies, for functions that have subfiles, whether the + sign or More. . . displays in the lower right location of the subfile indicating that the subfile contains more records.

- If P is specified, a + sign indicates that the subfile contains more records. This is the shipped default.
- If T is specified, More. . . indicates that the subfile contains more records. Bottom displays to indicate that the last subfile record is displayed.

Dynamic Program Mode

This option specifies whether the function automatically determines the initial mode of execution (add or update). This is based on whether records are present in the file. If there are any restrictor parameters or selection criteria, the records present are checked against the criteria.

- If N is specified, the initial program mode is fixed
- If Y is specified, the initial program mode is set dynamically

Exit After Add

This option specifies whether the function exits after addition of a new record.

Exit After Add is available only on Edit Record (EDTRCD, EDTRCD2, and EDTRCD3) functions.

- If Y is specified, you exit the program after successfully adding a record. You can use this option when the EDTRCD function is called from another function
- If N is specified, you do not exit the program after adding a record, except when Bypass Key Screen = Yes and the key is defined as an Input Restrictor parameter

Repeat Prompt

This option specifies whether the prompt redisplays after user processing of accepted prompt values.

Repeat Prompt is available only on Prompt Record (PMTRCD) functions that have validation of prompt and user data.

- If Y is specified, the prompt redisplays
- If N is specified, the prompt does not redisplay

Bypass Key Screen

If all key fields are supplied as restrictor parameters, this option specifies whether the key screen is bypassed (not displayed) before the detail panel . The Bypass Key Screen function option is available only on Edit Record (EDTRCD, EDTRCD2, and EDTRCD3) functions.

- If Y is specified and all key fields are non-blank at function execution time, the key screen is bypassed
- If N is specified, the key screen is not bypassed

Notes:

- Whenever the full key is passed into any EDTRCD, the key screen is bypassed even when Bypass Key Screen is N. The key fields do not need to be restrictor parameters for this to happen.
- If Bypass Key Screen is specified, all key field values must be supplied as restrictor parameters to bypass the display of the key screen. If key field values are not supplied as restrictor parameters, the key screen displays even though this option is Yes.

Post Confirm Pass

This option specifies whether the function is to re-read the database file and to update the subfile after confirmation; for example, to calculate line values based on totals.

If post confirm pass is specified, an additional user point is added to the function.

- If Y is specified, the function carries out a post confirm pass of the subfile
- If N is specified, the function does not carry out a post confirm pass of the subfile

Send All Messages Option

This option specifies whether an error message is sent to the message subfile at the bottom of the panel for the first error found, or for each error found. In either case, any outstanding messages are cleared each time Enter is pressed.

- If Y is specified, send all error messages to the message subfile at the bottom of the panel.
- If N is specified, send only the first error message.
- If M is specified, use the value of the YSNDMSG model value. Valid model values for YSNDMSG are *YES and *NO.

Exit Control

The exit control function options determine the execution characteristics of a program, such as:

- Whether or not it terminates or remains invoked but inactive
- Whether or not it reclaims resources as it terminates
- Whether messages are copied back to the calling program on termination

Reclaim Resources

This option specifies whether the i OS Reclaim Resources (RCLRSC) command is to be invoked when the program completes execution.

The command closes down any other programs and/or files that have been called and/or opened by the program, and reallocates their storage.

Reclaim Resources is valid only on external functions (functions implemented as programs in their own right). It is ignored for functions implemented in COBOL since this command is not valid for COBOL programs.

- If Y is specified, reclaim resources are invoked.
- If N is specified, reclaim resources are not invoked.

Closedown Program

This option specifies whether the RPG Last Record Indicator is set on when the program finishes execution.

- If Y is specified, all files are closed and the program is shut down
- If N is specified, all files remain open and a subsequent call is faster and performs a full program initialization

Notes:

- In either case, all internal variables are initialized to blanks and zeros on each call. If closedown is N, arrays are not cleared. This permits arrays to be used to store WRK variables. The PGM context variable *INITIAL CALL is available to determine if this is a first time subsequent call.
- COBOL has no direct equivalent of the RPG Last Record Indicator. The top-level program initiates the Run Unit. All programs called from this top program remain in the Run Unit until the top-level program itself closes down.

Copy Back Messages

This option specifies whether any messages outstanding on the program's message queue are copied to the previous program's message queue when the program terminates. The default value for new functions is specified by the YCPYMSG model value.

- If Y is specified, messages are copied back to the calling program's message queue.
- If N is specified, messages do not copy back to the calling program's message queue.
- If M is specified, the model default is used. Valid model values for YCPYMSG are *YES and *NO.

Commitment Control

These function option values determine the commitment control regime for a program.

Using Commitment Control

This option specifies whether the program that implements the function runs under i OS Commitment Control and, if so, whether it contains the main commit points. i OS commitment control provides a means of automatically grouping a number of database updates into a single transaction for the purposes of recovery: either all or none of the updates take place.

If you link together several functions as one transaction group, CA 2E determines where the commit points are located.

- If M (*MASTER) is specified, the program runs under commitment control. This program is the controlling program and contains the commit points. The program ensures that commitment is active by calling a CA 2E supplied program, Y2BGCTL. It also includes the appropriate commit points.
- If S (*SLAVE) is specified, the program runs under commitment control. No automatic start or commit points are included. You can add commit points by using the COMMIT built-in function. Commit operations can be performed by a calling program (typically *MASTER) function.
- If N (*NONE) is specified the program does not run under commit control.

Note: Any physical (PHY) file updated by programs running under commitment control must be journaled.

For more information about commitment control and journaling files, see the *i OS Programmers Guide.*

Exception Routine

The routine determines how program exceptions (errors) are handled for a program. This option applies only to RPG. It is not supported by COBOL/400.

Generate Exception Routine

This option specifies whether code for an exception handling routine (*PSSR) should be generated in the program that implements the function. This provides an opportunity for you to add user-defined exception handling. The default value for new functions is specified by the model value YERRRTN.

- If Y is specified, source code is generated that implements an error handling routine. In this case, all files in the program open explicitly using the OPEN operations. The *PSSR routine contains a call to the CA 2E supplied program, Y2PSSR. The source for this program is in QRPGSRC in Y2SYSRC and can be adjusted to supply specific error handling.
- If N is specified, source code is not generated for an error handling routine.

Generation Options

The generation options determine the generation mode and panel text constants for the program.

Generation Mode

This option specifies the method of database access used for the functions. Generation mode is determined by the model value YDBFGEN. You can override this value at the function level.

- If D is specified, the database access method is DDS.
- If L is specified, the database access method is DDL.
- If S is specified, the database access method is SQL.
- If A is specified, the access path generation value of the primary access path of the function is used.
- If M is specified, the value of the model value YDBFGEN is used. Valid model values for YDFGEN are *DDS, *SQL and *DDL.

Generate Help

This option specifies whether help should be generated.

- If Y is specified, help text is generated for this function.
- If N is specified, no help text is generated for this function.
- If O is specified, only generate help text for this function; do not generate any of the function's other components. You can only specify O for functions with Help Text for NPT set to U (*UIM).
- If M is specified, the value of the model value YGENHLP determines whether Help is generated for this function. Valid model values for YGENHLP are *YES, *NO, and *ONLY.

Help Type for NPT

This option specifies the type of help text associated with this function when it is generated as an NPT function.

- If T is specified, the help text is Text Management (TM). Help text is created in a source member and processed by the Display Help program.
- If U is specified, the help text consists of links from the Display File source to an i OS Panel Group compiled from source containing the i OS User Interface Manager (UIM) tag language.
- If M is specified, the model value YNPTHLP determines the type of help text generated. Valid model values for YNPTHLP are *UIM and *TM.

Generate as a Subroutine

This option specifies, for the EXCINTFUN type, whether to implement the function inline or as a subroutine.

- If Y is specified, the EXCINTFUN is implemented as a subroutine.
- If N is specified, the EXCINTFUN is implemented inline. This is the default.

Share Subroutine

This option specifies whether the generated source for an internal function (subroutine) is to be shared. This applies to CHGOBJ, CRTOBJ, DLTOBJ, RTVOBJ, and EXCINTFUN functions types.

- If Y is specified, generated source for subroutines is shared. In other words, source code is generated the first time an internal function is called and the source is reused for all subsequent calls to the function. The interface for the subroutine is externalized.
- If N is specified, source code is generated each time an internal function is called. The interface for the subroutine is internal.

Screen Text Constants

This option specifies the generation mechanism used for screen text.

- If L is specified, panel literals are hard coded in the device source.
- If I is specified, panel literals are placed in a message file. For the iSeries, they are accessed through the DDS MSGID keyword.
- If M is specified, the value of the YPMTGEN model value is used. Valid model values for YPMTGEN are *OFF, *LITERAL, and *MSGID.

Execution Location

This option specifies where the function is executed. This option is only valid for EXCINTFUN and EXCUSRPGM.

- If S is specified, execute the internal function or user program on the server
- If W is specified, execute the internal function or user program where the user point is found

Overrides if Submitted Job

This option specifies the source of SBMJOB parameter overrides when you submit a job for batch execution from within an action diagram. This option applies only to EXCEXTFUN, EXCUSRPGM and PRTFIL function types.

- If * is specified, use the default overrides defined by the '*Sbmjob default override' message attached to the *Messages file in Y2USRMSG
- If F is specified, use the override defined for the function
- This feature does not support function calls that contain multiple-instance array parameters.

Environment

The environment options determine the environment in which source code is generated for the program.

Workstation Implementation

This option specifies whether interactive CA 2E functions are to operate on non-programmable terminals (NPT) or on programmable workstations (PWS) communicating with an iSeries host. For programmable workstations, you also specify the PC run-time environment.

- If N is specified, the generated code operates on a non-programmable terminal (NPT) attached locally to the host computer.
- If G is specified, CA 2E functions are generated for non-programmable terminals together with a Windows executable running in a Windows environment under emulation to the host.
- If J is specified, CA 2E functions are generated for non-programmable terminals together with a Windows executable running in a Windows environment under emulation to the host and a Java executable running in a Windows environment using a Web browser with emulation to the host.
- If V is specified, CA 2E functions are generated for non-programmable terminals together with a VisualBasic executable running in a Windows environment under emulation to the host.
- If M is specified, use the value of the YWSNGEN model value to determine the type of workstation. Valid model values for YWSNGEN are *NPT, *GUI, *JVA, *VB.

Note: The values *GUI (G), *JVA (J), and *VB (V) require an interface to GUI products.

Distributed File I/O Control

This option specifies the kind of I/O control to generate for the function. This enables you to use DRDA to access files on multiple remote relational databases (RDBs). This option only applies if the function is generated using SQL or DDL for the Generation Mode.

Note: SQL access can be used in Generation Mode even if the access paths are generated using DDS.

- If S is specified (Synon Control), DRDA is used. The function is driven by the configuration entries (RDBs) of the function's default Retrieval access path. The table of the distributed files is created by executing the YCVTDSTFIL command. The configuration entries are added/modified using the YWRKDSTFIL command.

  Note: Synon Control is not applicable for Print files and Execute External functions. When Synon Control is specified for these two functions, User Control is used.

- If U is specified, (User Control), DRDA is used. The function contains distributed functionality/ capabilities but is not automatically driven by the configuration table entries. The initial relational database that the application is connected to are the current relational database unless overridden by action diagram logic that modifies the PGM context field *Next relational database.

  This field can be used within Synon Control to override the default processing.

- If N is specified, do not generate any distributed functionality for this function. This is equivalent to *NONE for the YDSTFIO model value.

- If M is specified, use the value of the YDSTFIO model value to determine the type of distribution I/O control access. Valid model values for YDSTFIO are *NONE, *USER, and *SYNON.

For more information on DRDA, see Generating and Implementing Applications in the chapter "Distributed Relational Database Architecture."

Null Update Suppression

This option specifies whether the CHGOBJ function type suppresses the record update when the before and after images of the record are the same. Use this function option to override the YNLLUPD model value.

- If N is specified, CHGOBJ always updates the record.
- If Y is specified, CHGOBJ checks whether to suppress database update both after the After Data Read and after the Before Data Update user points. The record is updated if the before and after images of the record differ.
- If A is specified, CHGOBJ checks whether to suppress database update after the After Data Read user point. The record is updated if the before and after images of the record differ.
- If M is specified, use the YNLLUPD model value to determine whether CHGOBJ is to update the record if the before and after images are the same. The valid model values for YNLLUPD are *NO, *AFTREAD, and *YES.

For more information about:

- Null update suppression see *Record Data Changed PGM Context, in the chapter "Modifying Action Diagrams"
- The CHGOBJ function, see CHGOBJ Database Function in the chapter, "Defining Functions"

Identifying Standard Header/Footer Function Options

dWhen you create a function with a device design, CA 2E assigns a default standard header/footer. You can override this default for any function by pressing F5 from the Edit Function Options panel to display a selection list of all standard header/footer functions.

The default standard header/footer is determined by the function options defined for the functions in the standard header/footer shipped file.

For more information and a list of the standard header/footer functions, see Standard Headers/Footers in the chapter "Modifying Device Designs."

To view the function options for the standard header/footer functions, follow these steps.

1. At the Edit Database Relations panel, type *S in the object field to display the list of shipped files beginning with those that begin with S.
2. Type **F** next to the *Standard header/footer file to display the list of all standard header/footer functions for the file. Each function contains function options and a header and footer format for a function panel design.
3. Type **Z** next to the standard header/footer function you want to view.
4. Press **F7** to display the Edit Function Options panel for the selected function.

Standard Header/Footer Function Options

The function options defined for each standard header/footer function apply to the functions to which the standard header/footer function is assigned.

CA 2E ships predefined standard header/footer functions, but you can also create and customize your own. The easiest way to do so is to make a copy of one of the standard header/footer functions and modify the copy.

132 Column Screen

This option specifies whether the display terminal, at which the display is shown, is 132 characters wide.

- If Y is specified, the terminal supports 132-character displays
- If left Blank, the terminal supports 80-column displays

Note: For a device function to allow 132 columns, its standard header/footer must have this option set to Y.

Enable Selection Prompt Text

This option specifies whether a default prompt message should appear on the device design for function keys and subfile selection text.

- If 1 is specified, a one-line prompt message appears for both function keys
- If 2 is specified, a two-line prompt message appears for both function keys
- If left Blank, the prompt messages are absent

Allow Right to Left/Top to Bottom

This option specifies whether bi-directional support is incorporated in the function.

- If Y is specified, cursor movement for input-capable text fields is right to left and top to bottom on the panel
- If left Blank, cursor movement is from left to right and top to bottom in input-capable fields

Function Options for Setting Header/Footer Defaults

The following function options determine the type of header/footer that is defined and the implicitly- selected default for that header/footer type.

- Use as default for functions
- Is this an Action Bar (Y), and the Default (D)
- Is this a Window (Y), and the Default (D)

The implicitly-selected default header/footer assigned to your functions is indicated on the Edit Function Options panel as follows.

```
¬ Implicitly set by mdl default
```

If you assign another standard header/footer function to your function, it is indicated on the Edit Function Options panel as follows.

```
¬  Explicitly selected
```

Note: CA 2E may automatically change implicitly-selected header/footers if you change the YSAAFMT model value or the YWSNGEN model value.

For more information on implicitly-selected header/footers, see Design and Usage Considerations and the Examples later in this chapter.

Use As Default for Functions

This option applies only if the model value YSAAFMT is *CUAENTRY and the model value YWSNGEN is set to *NPT. It specifies whether this standard header/footer is assigned to functions as the implicitly-selected default.

- If Y is specified, this standard header/footer is assigned to functions as the implicitly-selected default
- If left Blank, this standard header/footer is not assigned to functions as the implicitly-selected default

Is This an Action Bar

This option specifies whether the function to which this standard header/footer is assigned contains an action bar.

- If Y is specified, the function contains an action bar.
- If N is specified, the function does not contain an action bar.
- If D is specified, the function contains an action bar. In addition, if the YSAAFMT model value is set to *CUATEXT, all panel-based function types other than SELRCD has this standard header/footer assigned as the implicitly-selected default.

Is This a Window

This option specifies whether the functions to which this standard header/footer is assigned contains a window.

- If Y is specified, the function contains a window.
- If N is specified, the function does not contain a window.
- If D is specified, the function contains a window. In addition, if model value YSAAFMT is set to *CUATEXT, SELRCD functions has this standard header/footer function assigned as the implicitly-selected default.

Design and Usage Considerations

Following are points to consider if you want to customize or change the way CA 2E assigns standard header/footers for your functions.

- A standard header/footer function can be an action bar, a window, or neither. It cannot be both an action bar and a window.
- To create a standard header/footer function with neither an action bar nor a window, set both the following function options to N.
  - Is this an Action Bar
  - Is this a Window
- There can be only one default standard header/footer function for each header/footer type. In other words,
  - Only one standard header/footer function can have the Use as default for functions option set to v.
  - Only one standard header/footer function can have the Is this an Action Bar option set to D.
  - Only one standard header/footer function can have the Is this a Window option set to D.

  If you set a new standard header/footer to be a default, CA 2E automatically resets the corresponding function option for the previous default header/footer function so it is no longer the default.
- All existing functions that have implicitly- selected header/footers are always assigned to the default header/footer. If you modify the default header/footer, CA 2E immediately reassigns the implicitly-selected header/footers. This also occurs if you change the YSAAFMT model value and may occur if you change the YWSNGEN model value or the Workstation Implementation function option for a function. See the examples at the end of this topic.

  To prevent this, you can explicitly select the same or another standard header/footer for any function using the Edit Function Options panel.

Examples

Suppose the YSAAFMT model value is set to *CUAENTRY and the YWSNGEN model value is set to *NPT. In addition, suppose the standard header/footer function options are set as follows:

Standard Header/Footer Functions	Use as default for function options	Is this an Action Bar option?	Is this a Window option?
Header/Footer1	Y	N	N

Standard Header/Footer Functions	Use as default for function options	Is this an Action Bar option?	Is this a Window option?
Header/Footer2	blank	D	N
Header/Footer3	blank	N	D

The following examples all refer to this basic scenario.

Example 1

With these settings, all functions created have Header/Footer1 specified as the implicitly-selected default.

Example 2

If you change the YSAAFMT model value to *CUATEXT, all functions other than SELRCD have Header/Footer2 assigned as the implicitly-selected default. SELRCD functions will have Header/Footer3 assigned as the implicitly-selected default.

Example 3

If you change the YSAAFMT model value to *CUATEXT, create a new Header/Footer4, and set it to be the default action bar, the standard header/footer function options changes as follows:

Standard Header/Footer Functions	Use as default for function options	Is this an Action Bar option?	Is this a Window option?
Header/Footer1	Y	N	N
Header/Footer2	blank	Y	N
Header/Footer3	blank	N	D
Header/Footer4	blank	D	N

Note: CA 2E has automatically reset the Is this an Action Bar option for Header/Footer2 to Y. All functions other than SELRCD have Header/Footer4 assigned as the implicitly-selected default. SELRCD functions still have Header/Footer3 assigned as the implicitly- selected default.

Chapter 8: Modifying Function Parameters

This chapter identifies the basic properties and the roles of function parameters and explains how to define them for functions. This chapter also explains how to use arrays as parameters.

This section contains the following topics:

Understanding Function Parameters

Function parameters specify which fields can be passed between the calling and the called functions. Each call can pass different values in these fields, but the definitions of the fields themselves remains the same. You assign the parameter roles, which direct the function to use that parameter m in a specific way.

Identifying the Basic Properties

Parameters have the following four basic properties:

- Name
- Usage
- Role
- Flag error status

Name

Function parameters are defined by reference to the field from which they receive or to which they return a value.

Usage Type

A parameter's usage definition determines how the parameter is allowed to be used. Parameters can be used in one of four ways depending on how they are received from or returned to the function. The direction of movement is always viewed from outside the function whose parameters are being defined. The usage types are as follows:

Input Only

A value is passed into the function when the function is called, but the function does not change this value and the same value is returned.

Output Only

A value is returned from the function for the parameter when the function completes. Any initial value passed in this variable is set to blank or zero at the start of the function.

Both (Input/Output)

A value is passed into the function for the parameter when the function is called and a value, possibly different, is returned to the calling function when the function completes processing.

Neither

No value is passed into the function for the parameter nor is a value returned for the parameter when the function ends. Neither parameters are available for use as local variables within the function.

Note: Neither parameters are, in some instances, preferable to WRK context variables. WRK context variables are global to the function and can be updated inadvertently from internal functions within the main external function. The local nature of Neither parameters avoids this potential problem.

The following table shows the types of function parameter usage.

Parameter type	Passed in	Returned	MAP
Input	Y	N	Y
Output	N	Y	N
Input/Output	Y	Y	Y
Neither	N	N	Y

Flag Error Status

Flag error status specifies whether a calling function should indicate that an error, which occurs in the called function, is associated with the parameter field. If so, the field is highlighted on the display of the calling function if the parameter field in the called function returns a non-blank return code.

Flag error status applies only to the following function types:

- SNDERRMSG (send error message)
- EXCMSG (execute message)
- Any external function

Send Error Message specifies that an error message be sent to a calling function. The Send Error Message function is attached to a CA 2E shipped file called * MESSAGES.

For more information on function types and external functions, see the chapter "Defining Functions."

By default, all parameters of the called function that ended in error are highlighted if they appear on a display. You can override this to suppress error flagging for a parameter by altering the default attributes for fields in error on the Edit Screen Field Attributes panel.

Identifying Default Parameters

Certain standard functions have predefined default parameters. When you create any of the function types, the appropriate default parameters are automatically created.

Function	Default Parameters	Usage
CHGOBJ	All fields from update index of based-on file	I
DLTOBJ	Key fields from update index of based-on file	I
RTVOBJ	Key fields from the associated access path	I
SELRCD	Key fields from the update index of based-on file	B

Key: I = Input Only Usage
B = Both Input and Output Usage

Identifying the Return Code

All standard function types other than EXCUSRPGM and EXCUSRSRC have an implicit parameter, the return code. The return code is used to inform the calling program of the circumstances under which the called program is exited. The return code is not shown on the Edit Function Parameters panel but is automatically declared in the generated source code as the first parameter.

Therefore, when calling your application program from a menu or a command line you must always specify a parameter for the return code. This parameter must be the first parameter.

For example, when calling a function from a command line:

```
CALL  ABCDEFR ' '
```

Note: You can also use the Call a Program (Y2CALL) command to call the function. This command is especially useful if the function's parameter interface is complex or has changed. It determines the parameters, including the return code, required by a function directly from details contained in the model. You can provide values for all input-capable fields and you can reuse these values for subsequent calls.

For more information on the Y2CALL command, see the *Command Reference Guide*.

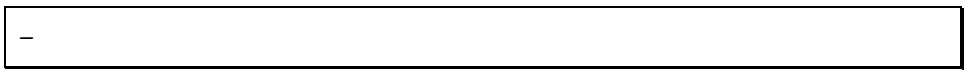
You can retrieve or change the value of the return code parameter within the action diagram of a function by referencing the PGM context field *Return Code. This field can be set by CA 2E to one of its defined conditions, such as, *Record not found. The conditions that are supplied for the *Return Code field are:

- *Data update error
- *Normal
- *Record already exists
- *Record does not exist
- *Substring error
- *User Quit requested

You can test against any of these supplied conditions or you can add other conditions to the list.

The best way to check for not equal to *NORMAL can be done using a CASE condition with an *OTHERWISE or by using a compound condition.

For example:



```
. .-CASE
.  -PGM.*Return Code is *NORMAL
.  Print Customers - Customer *
.  -*OTHERWISE
.  Print Customers Credit - Customer *
.  -ENDCASE
;—
```

Understanding the Role of the Parameter

The role of a parameter specifies how the parameter is used in the function into which it is passed. Each category of parameter role applies to certain standard function types. The parameter roles are as follows:

Map Parameter

This parameter is automatically moved to a corresponding field on the receiving function's panel design. If the field does not exist on the device design, it is added to it. Specifying fields as mapped Neither parameters is a way of adding fields to a panel design without the need for passing them into the function. The Map option is ignored for reports and functions that do not have an associated device design.

Note: If you make a change to the parameter entry on the Edit Function Parameters Detail panel, the entry defaults to the Map role.

Restrictor Parameter

This parameter is used to restrict the records from a database file that can be displayed, changed, or printed by the function. Restrictor parameters must be key fields on the access path to which their function attaches and can only be used hierarchically; that is, major to minor key sequence.

A minor key can only be a restrictor parameter if all keys major to it are also restrictor parameters. Restrictor parameters are automatically mapped and default to output on the panel, but may be changed to input on DSPFIL, SELRCD, and EDTFIL function types.

For example, a function that displays a list of records (such as DSPFIL) could allow the user to select a particular record with a line selection option. The keys of the selected record could be passed as restrictors to another function (such as EDTRCD). The called function then process only the selected record.

Using Restrictor Parameters

The following examples show the effect of using restrictor parameters on single-record display styles and on multiple record display styles.

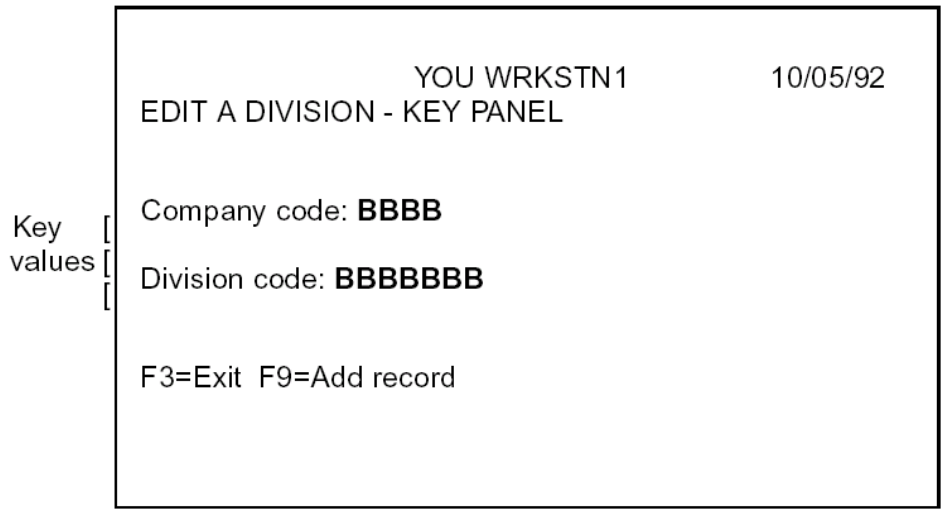
For example, if a Division is defined by the following relations:

```
FIL  Company   REF  Known by   FLD  Company code     CDE
FIL  Company   REF  Has        FLD  Company name     TXT
```

```
FIL  Division  REF  Owned by   REF  Company          FIL
FIL  Division  REF  Known by   FLD  Division code     CDE
FIL  Division  REF  Has        FLD  Division name     TXT
FIL  Division  REF  Has        FLD  No of employees   NBR
```

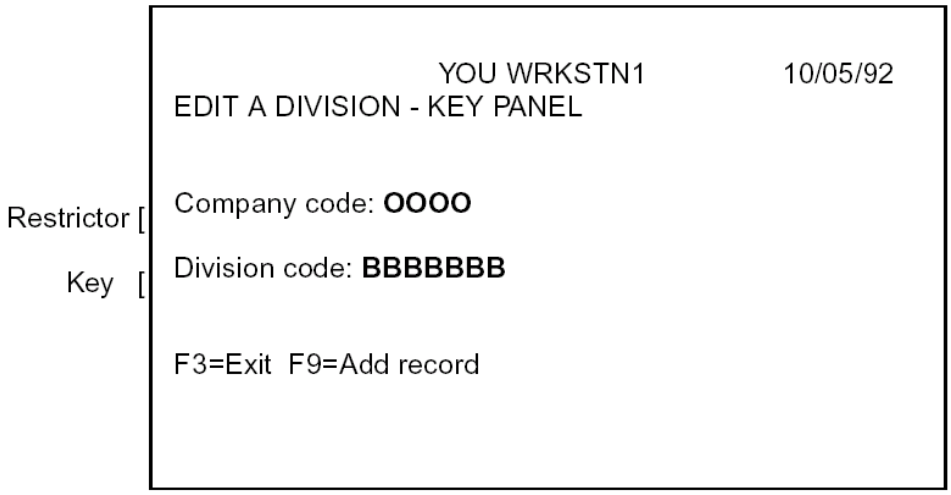
Single-Record Panel Design Without a Restrictor

If an Edit Record function is specified to edit the Division file without a restrictor parameter being declared, the default device design for the key panel would appear as follows. All of the key fields are input capable.



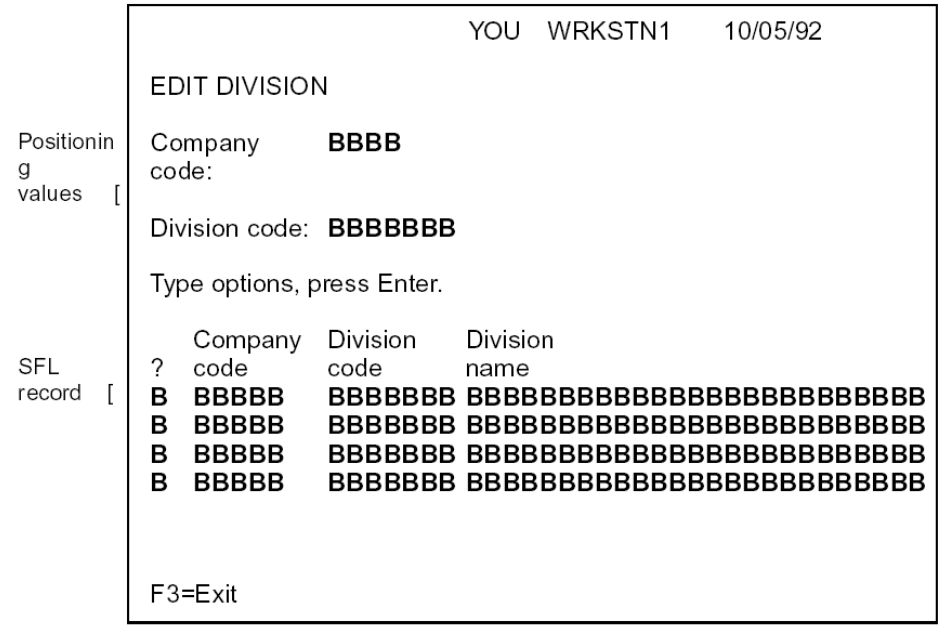
Single-Record Panel Design with a Restrictor

If the Company code is specified as a restrictor parameter, the Company code no longer appears as an input-capable field but becomes protected, as it is assumed that its value is being provided as an entry parameter:



Multiple-Record Panel Design without a Restrictor

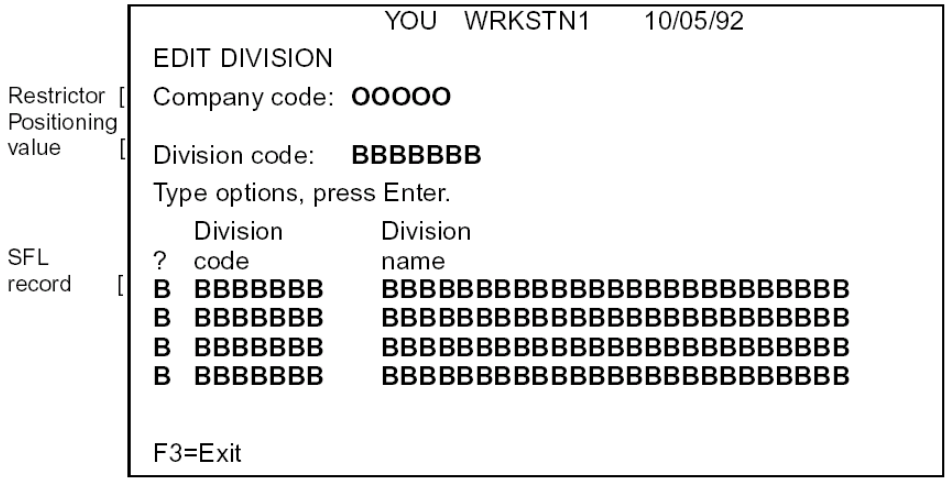
The effect of a restrictor on a multiple-record (subfile) panel design is similar. Consider that an Edit File function edits a Division file. If no restrictor parameter is specified, the default device design appears as follows:



Each subfile record contains all the fields from the underlying Division file.

Multiple-Record Panel Design with a Restrictor

If a Company Code field was specified as a restrictor parameter, the Company Code field no longer appears on each individual subfile record but would instead be shown on the subfile control record. The Company Code field is protected, as it is assumed that its value is being provided as an entry parameter.

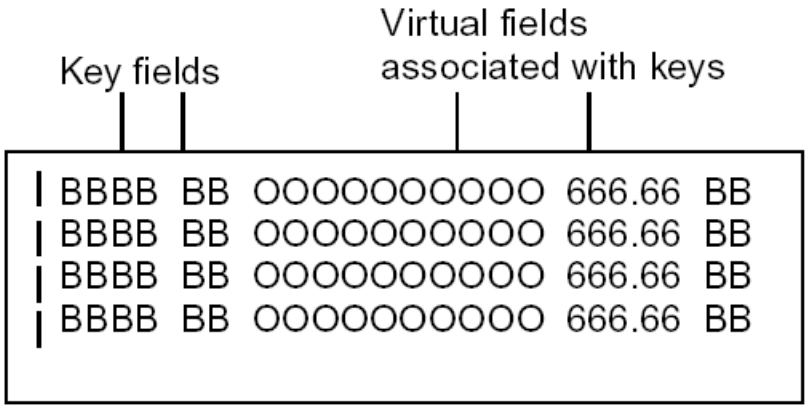


Note: You can override the restrictor on this panel to be input capable.

Virtual Fields and Restrictors on Subfiles

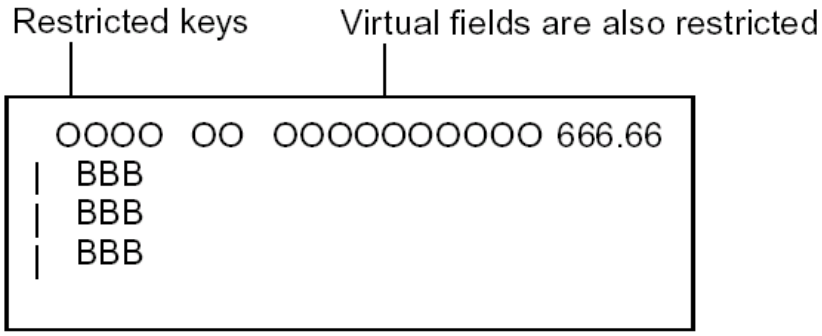
Some special considerations arise for subfile displays when there are virtual fields on the subfile record associated with a relation for which all of the keys are specified as restrictor parameters.

The following examples show the effect of restricting and not restricting a subfile's virtual fields onto the header format.

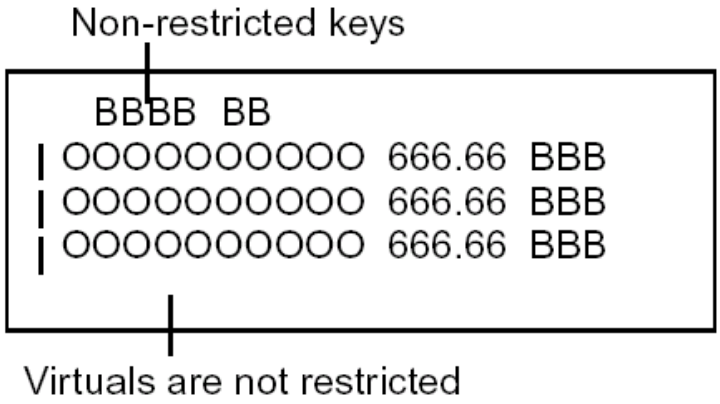


Different results are obtained if the virtual fields are present in the access path of the function due to the virtualization of fields that are virtualized in a related file. For example, Customer Name is a virtual on Order Header and is re-virtualized to Order Detail. Two possible differences are:

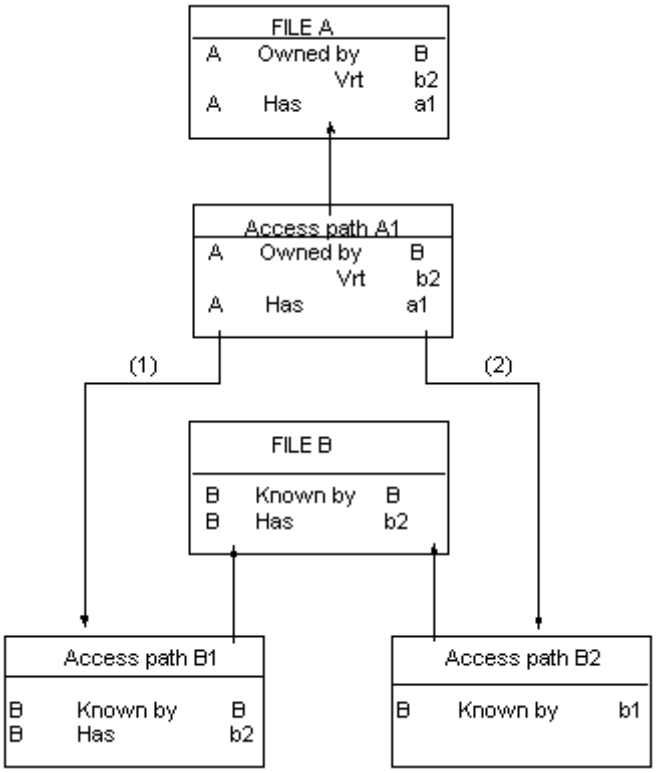
The virtual fields are restricted: They are removed from the individual subfile records and are placed on the subfile control with the restricted fields. This happens if the virtual fields associated with the keys also are restricted.



The virtual fields are not restricted: They remain on the individual subfile records. This happens if the fields are defined as virtual fields on the access path of the function file but are not present on the access path of the referenced file that is restricted.



The access path combinations for restrictor parameters are:



A1 to B1

Field b2 is present on both the referencing (A1) and referenced (B1) access paths. Virtual fields are restricted in functions based on access path A1.

A1 to B2

Field b2 is present on the referencing access path (A1), but not on the referenced access path (B2). Virtual fields are not restricted in functions based on access path A1.

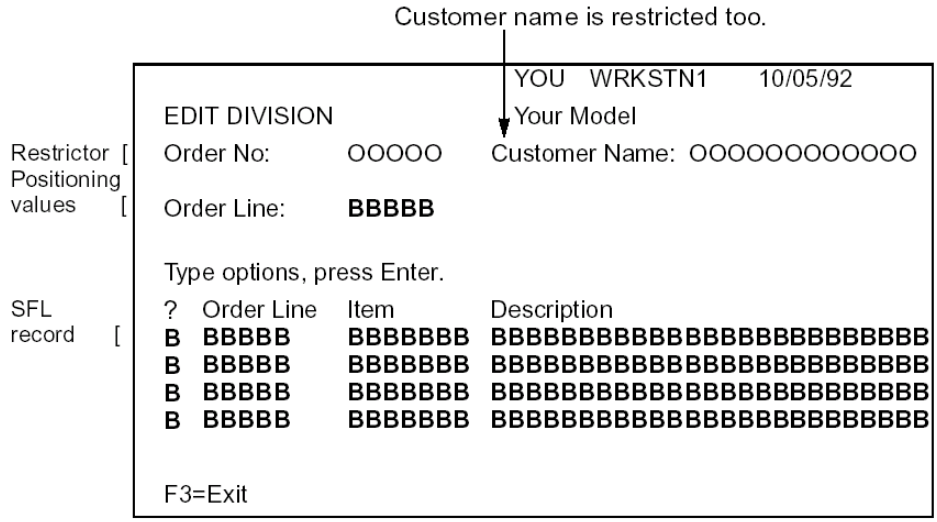
Example of Virtual Restrictor Usage

In the following example, the Customer Name is specified as a virtual field on the Order Header and is revirtualized to the Order Detail file.

```
Order Header        Known by        Order No
Order Header        Refers to       Customer
                    VRT             Customer Name
Order Detail        Known by        Order Line No
Order Detail        Refers to       Item
                    VRT             Item Description
                    VRT             Customer No
                    VRT             Customer Name
```

This example creates a device design over Order Detail, where Order No. is a restrictor as follows.

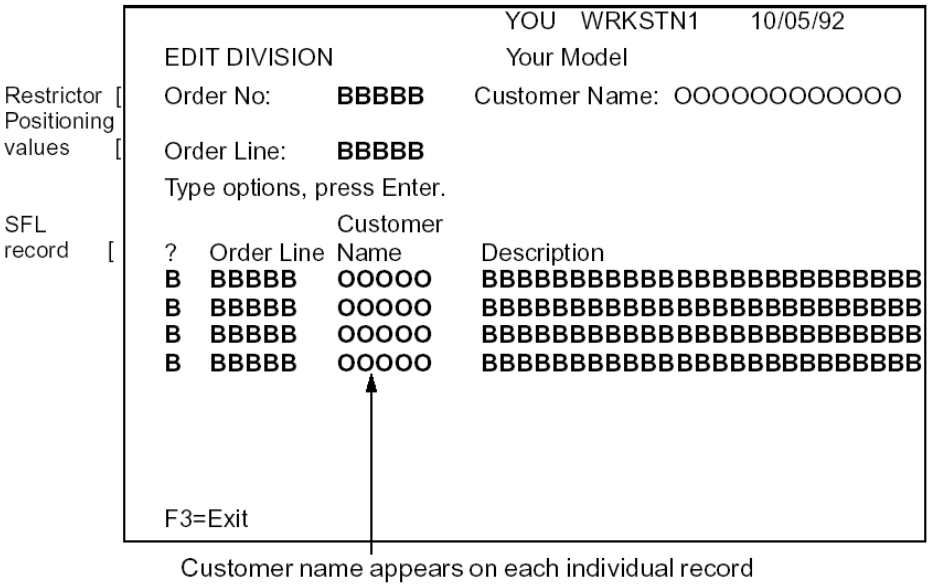
Device Design with Restricted Virtual Fields



If the Customer Name field is not present on the access path of the referenced file, Customer Name is no longer being included on the subfile control.

However, if Order No. is not a restrictor, the device design would be as follows.

Device Design Without Restricted Virtual Fields



Positioner Parameter

This parameter is used to position a function to start reading records from a database file at a particular record. Positioner parameters can be used by themselves or in conjunction with restrictor parameters. They must be key fields on the access path to which their function attaches and can only be used hierarchically.

For example, a minor key can only be a positioner parameter if all major keys to it are also positioner or restrictor parameters. Positioner parameters are automatically mapped and can be either Output or Both on the panel.

The following table is an example of the use of positioner parameters.

FIL	Company	REF	Known By	FLD	Company code	CDE
FIL	Company	REF	Has	FLD	Company name	TXT
FIL	Division	REF	Owned by	REF	Company	FIL
FIL	Division	REF	Known By	FLD	Division code	CDE
FIL	Division	REF	Has	FLD	Division name	TXT
FIL	Division	REF	Has	FLD	No of employees	NBR

If you define a PRTFIL function based on a retrieval access path over the division file (such as, with keys Company Code and Division Code fields) and if you want to specify selection you could either:

- Make both Company Code and Division Code positioner parameters in order to read all records starting at specified values for both key fields
- Make Company Code a restrictor but Division Code a positioner parameter in order to read all records for a specified Company file starting at a given division

Vary Parameter

This parameter can have a varying length. The vary parameter is useful when interfacing with user-written subroutines and programs. Domain checking is ignored.

Vary parameters are valid on functions that generate an external high-level language program, that is RPG or COBOL. You need to ensure that the parameters function properly if the domains do not match.

Allowed Parameter Roles

The following table shows the allowed parameter roles for the standard functions.

Function	Map	Restrictor (Keys Only)	Positioner (Keys Only)	Vary
PMTRCD	Y	Y	-	-
EDTRCD(1,2,3)	Y	Y	-	-
DSPRCD(1,2,3)	Y	Y	-	-
SELRCD	Y	Y1	-	-
DSPFIL	Y	Y1	-	-
EDTFIL	Y	Y1	-	-
EDTTRN	Y	Y	-	-
DSPTRN	Y	Y	-	-
PRTFIL	*	Y	Y	-
PRTOBJ	*	Y	Y	-
RTVOBJ	-	Y	Y	-
CRTOBJ	-	Y	-	-
CHGOBJ	-	Y	-	-
DLTOBJ	-	Y	-	-
EXCEXTFUN	-	-	-	Y
EXCINTFUN	-	-	-	-
EXCUSRSRC	-	-	-	Y
EXCUSRPGM	-	-	-	Y

Notes:

* Denotes that the map is allowed but does not add the parameter to the device design.

1 Indicates that you can override the restrictor parameter, which is normally output only, to be input capable.

Defining Function Parameters

To define a parameter for a function, specify the field that is passed to or from the function with one of these two procedures:

- Specify the parameters from the Edit Function Parameters panel.
- Specify the parameter in the action diagram when you specify the link between the functions.

Defining Parameters with the Edit Function Parameters Panel

The Edit Function Parameters panel defines the parameters that are passed to the function by the calling function. How the parameter is used in the called function depends on the called function's type or the processing specified by that function. Parameters can either be defined as a list of specific fields or as selections of fields from a list of access paths, arrays, or files.

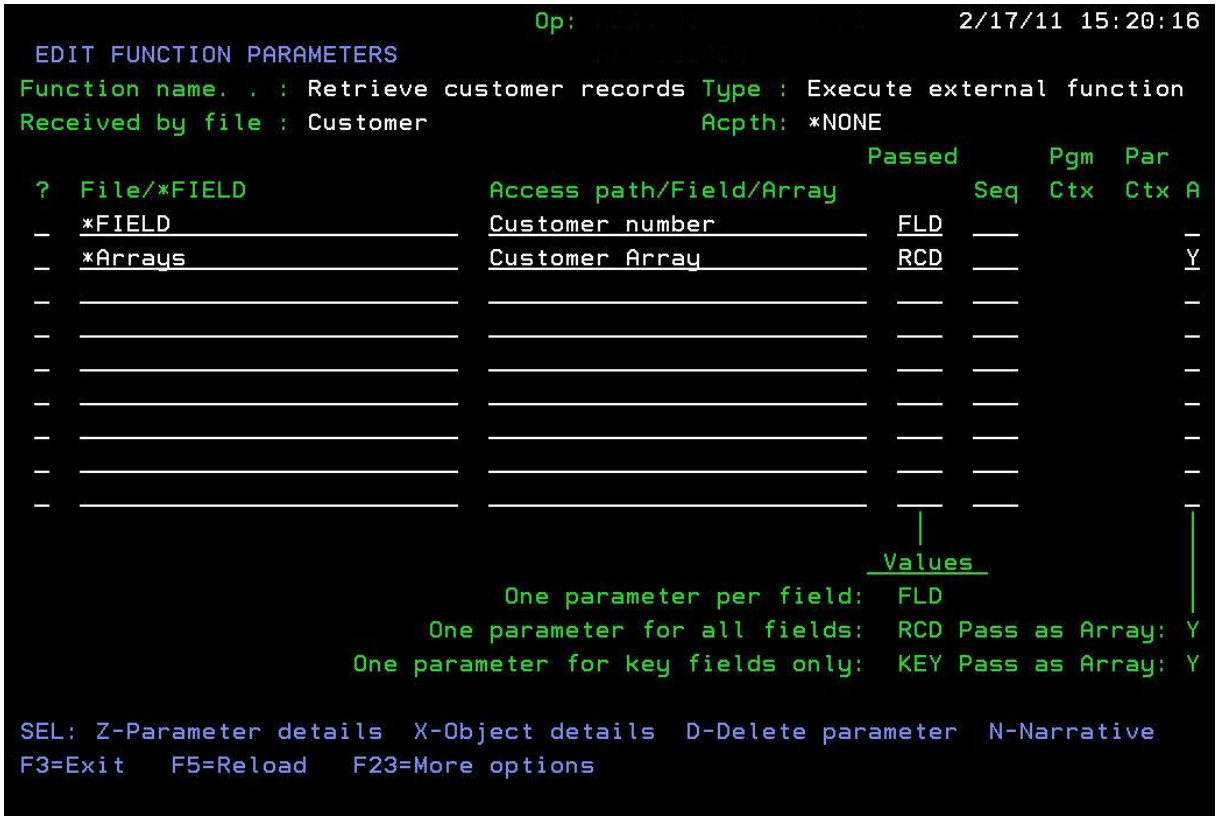
To access the Edit Function Parameters panel

1. View the functions. From the Edit Database Relations panel, type F next to the selected file, and then press Enter.

   The Edit Functions panel appears.

2. View the parameters, type P next to the selected function, and then press Enter.

   The Edit Function Parameters panel appears:



Use one of the following methods to specify the parameters:

To specify an individual field

1. Type *FIELD or *F in the File/*FIELD column.
2. Type the name of the field in the Access path/Field/Array column.

   By entering *?* this field can be used for prompting.

   The Passed field defaults to FLD.

To specify a set of fields from a file, access path, or array:

1. Type the name of the file in the File/*FIELD column. For arrays, type **Arrays* (or **A*).

   By entering ? this field can be used for prompting. A * defaults to the name of the file over which the function is built.

2. Type the name of the access path or array name in the Access path/Field/Array column.

   By entering ? this field can be used for prompting.

   Note: If you use *NONE as the access path then all fields, actual and virtual, are associated with the file are available for selection. This removes the need to tie the parameter entry to a specific access path and can reduce the impact of a change to the definition. This approach is particularly relevant when a subset of fields is selected.

3. Type the value that the field is passed as:

   FLD

   Each specified field is passed as an individual parameter. This must be specified for *FIELD or *NONE parameter lines.

   RCD

   A single parameter with the length of the specified access path is passed. The parameter contains space for all the fields associated with the access path, which can individually be specified as parameters.

   KEY

   A single parameter with the length of the combined keys of the specified access path is passed. The parameter contains space for all the key fields that can be individually specified as parameters.

4. Enter *Y* or leave a blank value for the A (Pass as Array) field.

   The following situations apply when using the A field:

   - If the function is not EXCEXTFUN or EXCUSRPGM, the field is not available.
   - The new ‘A’ (Pass as Array) field is available for all EXCEXTFUN and EXCUSRPGM parameters. However, if Y is specified for a parameter that is not an array based on the *Arrays file, or when the parameter is passed as FLD, then an error message is sent.
   - Y is only valid when the parameter is an array based on the *Arrays file, and only when the parameter is passed as RCD or KEY.
   - When you model in the action diagram of function A a call to function B, if a parameter is passed as an array on A it must be passed as an array on B.
   - No fields can be dropped on a parameter being passed as an array.
   - Though the usages of the subfields on a parameter passed as an array can be mixed, the usages must be compatible, such that the calling function can call the called function

5. Define the specific fields that are to be passed from the file, access path, or array on the Edit Function Parameter Details panel.
   - For both *FIELD and other parameter entries, define the role and usage of the parameter.
   - You can adjust the sequence of the entries by keying sequence numbers on each parameter line.

Non-unique Sequence Numbers

Sequence numbers do not need to be unique for a function where the Duplicate Parameters Option is set to *N*.

Identifying Functions with a Non-unique Parameter Sequence

Use the YCHKFUNPAR command to analyze a model and identify all functions that exhibit specific parameter interface problems. For example, when you have non-unique sequence numbers on the parameter interface of a function where the function option Duplicate Parameters is set as *Y*.

Resolving Function with a Non-unique Parameter Sequence Number

To rectify a function with a non-unique parameter sequence number

1. Access the EDIT FUNCTION PARAMETERS panel and change the sequence numbers to be unique.

```
                                  Op: COCSI01    QPADEV000J   4/14/11  8:13:53
 EDIT FUNCTION PARAMETERS             SBC368MDL
Function name. . : Verify account              Type : Execute external function
Received by file : account                     Acpth: *NONE
                                                      Passed      Pgm  Par
 ?  File/*FIELD              Access path/Field/Array                 Seq  Ctx  Ctx A
 _  account                  Retrieval index                  RCD     1  PR1  PR1 _
 _  account                  Retrieval index                  RCD     2  PR2  PR2 _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
 _  _____________________    _____________________________    ___   ___             _
                                                      Values
                                 One parameter per field:  FLD
                               One parameter for all fields:  RCD Pass as Array:  Y
                            One parameter for key fields only:  KEY Pass as Array:  Y

SEL: Z-Parameter details  X-Object details  D-Delete parameter  N-Narrative
F3=Exit   F5=Reload   F22=File locks   F23=More options
```

2. Access the Action Diagram of the function and ensure that all parameter contexts are modified appropriately to point to the correct parameter.

Invalid Duplicate Parameter fields

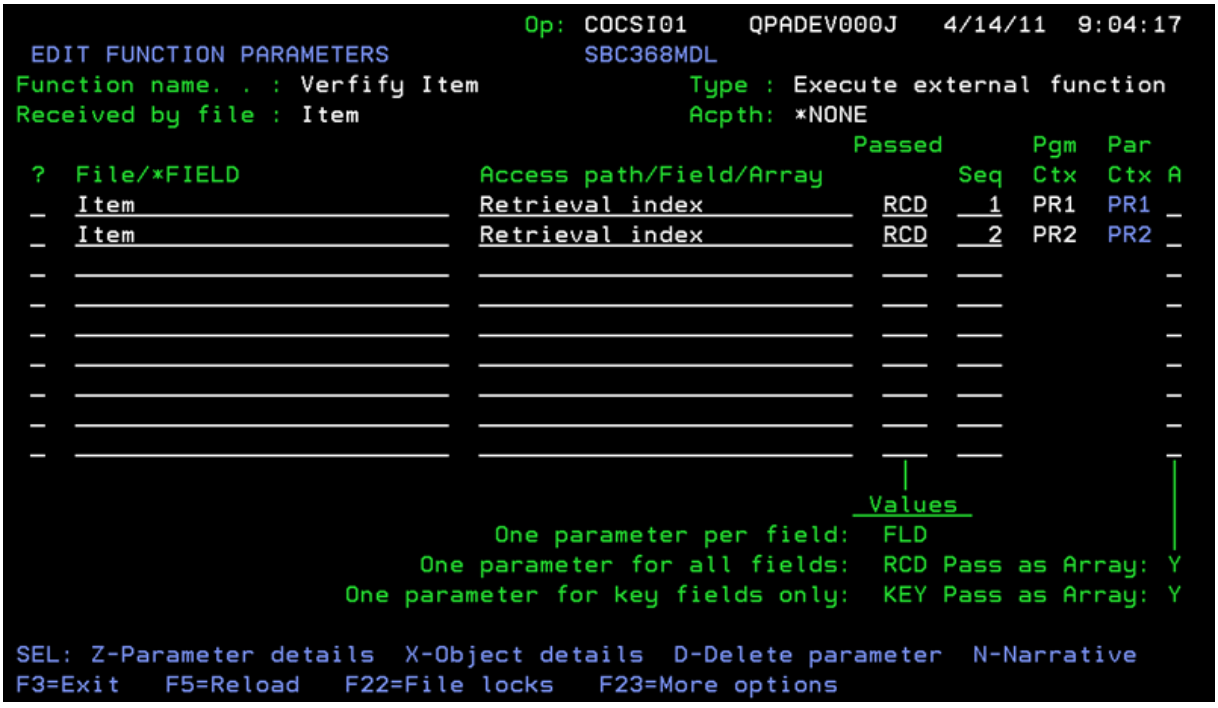
When a function has its Duplicate Parameters option set to *N*, each parameter field should be unique, regardless of the usage. The only exception is that a field can appear once for Input and once for Output.

There are two scenarios where a violation of this restriction and exception can occur:

- Earlier, the user was only warned after violating the change, rather than being prevented from the violation.
- Now it is possible to create a function with an invalid parameter interface. This can occur when you turn the DUPLICATE PARAMETERS function from *Y* to *N,* and any (previously valid) duplicate parameters are not modified appropriately.

In the following example, the function Verify Item has two identical parameters: Item, Retrieval index, passed as RCD with seq=1 (PR1) and seq=2 (PR2). The subfields on the Item file are: Item code, Item description, Item Price and Item barcode, which are all passed as Both on PR1 and PR2.

Because the Duplicate Parameters Option is set to Y, this is a valid operation:



If the Duplicate Parameters Option is changed from Y to N, then the parameter interface becomes invalid. However, you will not receive a warning or error message.

Identifying Functions with Invalid Duplicate Parameter Fields

Use the YCHKFUNPAR command to analyze a model and identify all functions that exhibit specific parameter interface problems. Identifying invalid duplicate parameter fields on the parameter interface of a function where Duplicate Parameters is set to *N* is one of the issues where YCHKFUNPAR is helpful.

Rectifying Functions with Invalid Duplicate Parameter Fields

To rectify a function with invalid Duplicate Parameter fields, you must modify the function so that it does not violate the restriction and exception.

When a function has the Duplicate Parameters option set to *N*, each parameter field should be unique, regardless of the usage. The only exception is that a field can appear once for Input and once for Output.

There are two approaches to rectify this situation:

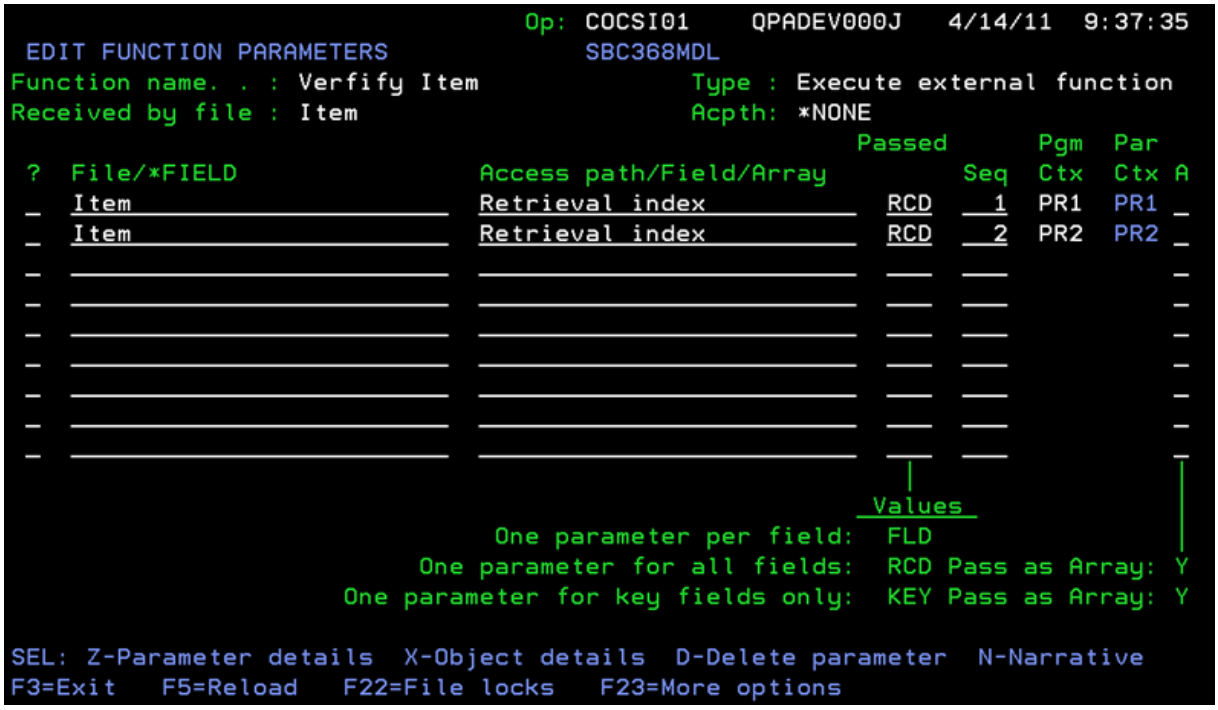
- Change the function option Duplicate Parameters to *Y*.

  Note: While this approach immediately makes the parameter interface valid, you must update any reference to PAR context in the action diagram to refer to the appropriate duplicate parameter PR1 through PR9 context.

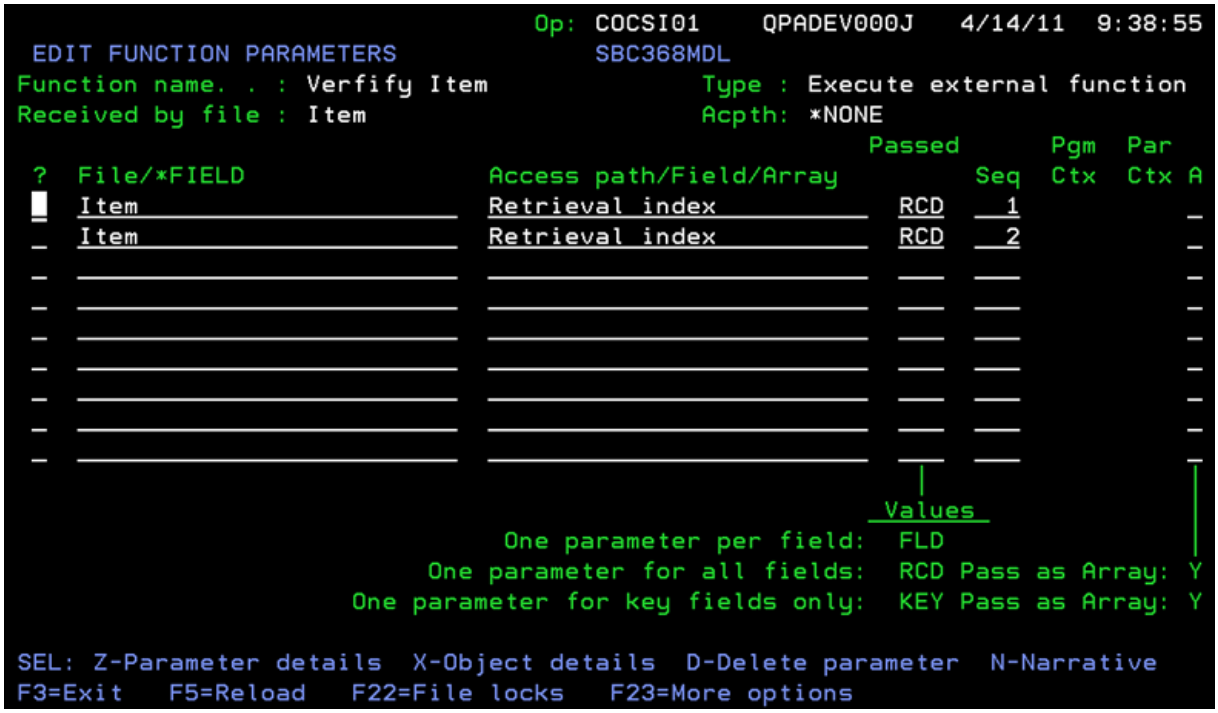
- Leave the Duplicate Parameters setting as is, but modify the usages of parameter fields so that the restriction and exception are not violated, which is detailed in the following examples.

To Rectify Functions with the Duplicate Parameters Option set to *Y* then set to N

1. Verify Item has Duplicate Parameters set to Y.

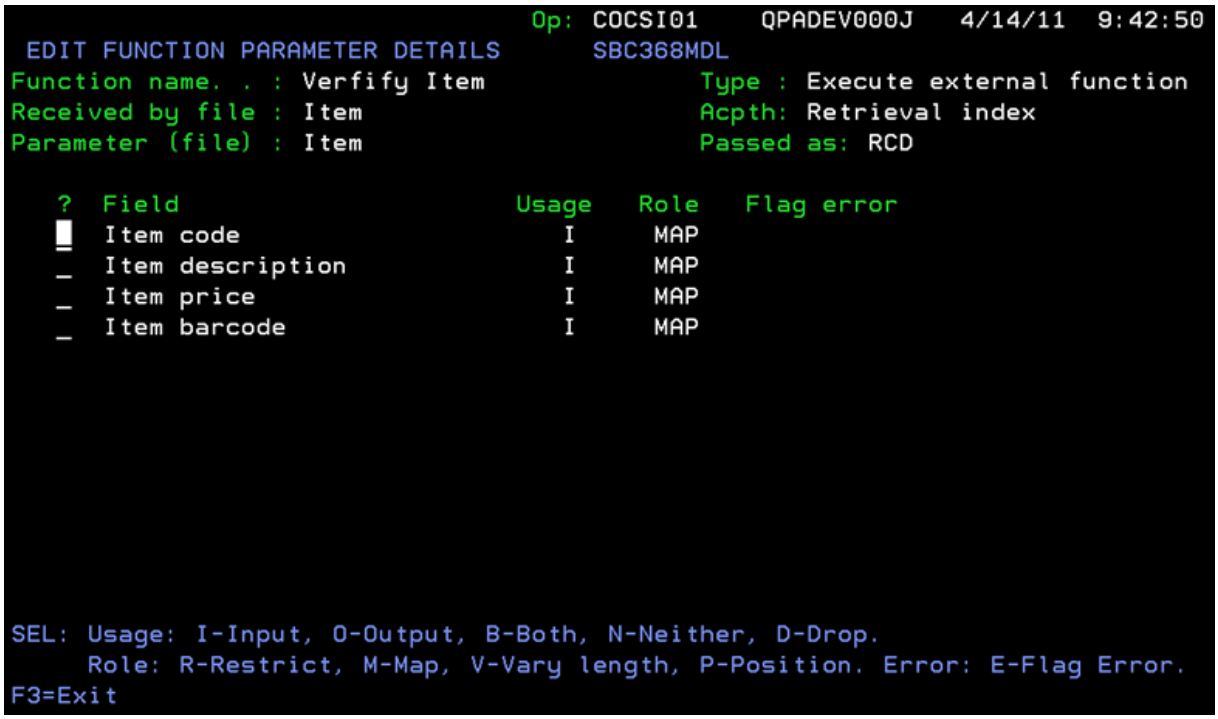


After the Duplicate Parameters function option has been changed to N, no warning is sent. However, the parameter interface is invalid because subfields appear on two different parameters with usage Both, as shown in the following example:

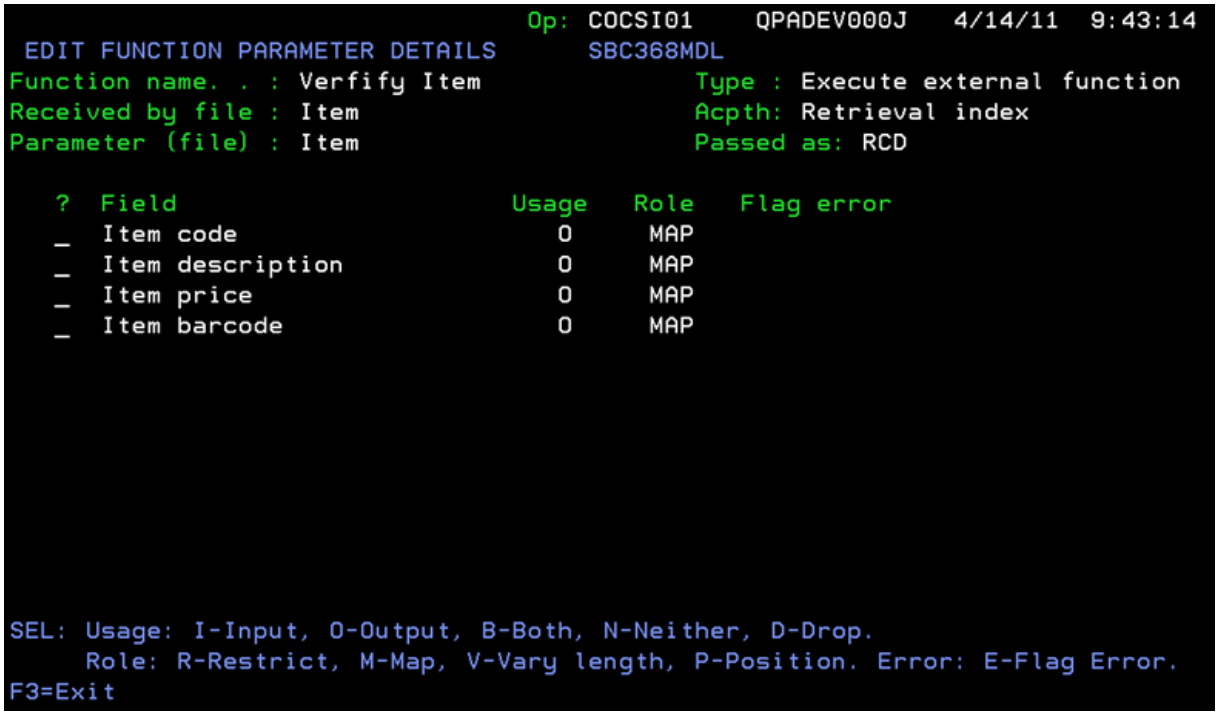


2. Complete one of these steps:
   - Remove the second parameter entirely.
   - Change the usages of the first parameter to input and the usages of the second parameter to Output.

3. Specify *Input* for all the fields on the first parameter:



4. Specify *Ouput* for all the fields on the second parameter:



This parameter interface is now valid.

5. Visit the action diagram and ensure that the parameter contexts are properly selected.

Defining the Parameter's Usage and Role

After you specify the parameter, you can define the parameter's usage and role on the Edit Function Parameter Details panel.

The following situations with the functions EXCEXTFUN and EXCUSRPGM apply to this panel:

- When the function is not EXCEXTFUN or EXCUSRPGM, or when the function is EXCEXTFUN or EXCUSRPGM but parameter is not passed as an array, RCD (ARRAY) or KEY (ARRAY), then Number of Elements is not available.
- When the function is not EXCEXTFUN or EXCUSRPGM, the *Passed as* field cannot have values of KEY (ARRAY) or RCD (ARRAY).
- When the function is EXCEXTFUN or EXCUSRPGM and the parameter is not passed as an array, then the *Passed as* field cannot have values of KEY (ARRAY) or RCD (ARRAY).
- When function is EXCEXTFUN or EXCUSRPGM, parameter is passed as an array(A='Y'), and the Edit Function Parameters panel (see page 271) indicates Passed=RCD, then the *Passed as* field is RCD (ARRAY).
- When function is EXCEXTFUN or EXCUSRPGM, parameter is passed as an array(A='Y'), and the Edit Function Parameters panel (see page 271) indicates Passed=KEY, then the *Passed as* field is KEY (ARRAY).
- When function is EXCEXTFUN or EXCUSRPGM and parameter is passed as an array (RCD or KEY), then the Number of elements displays the number of elements, as defined on the array being passed. You can view and modify the array's definition in the *Arrays file.

To define the parameter's usage and role

1. Zoom into the parameter, type **Z** next to the selected parameter, and then press Enter.

   The Edit Function Parameter Details panel appears:

Notes:

- If you entered a file, access path, or an array, those fields are listed on this panel. If the function has more than eight parameters, a parameter selection field displays in the header.
- When Passed as=RCD (ARRAY) or =KEY (ARRAY), Number of elements displays the Number of elements, as defined in the array being passed.

2. Enter the selected usage type in the *Usage* column, next to the parameter you are defining. The options are:
   - I (Input)
   - O (Output)
   - B (Both)
   - N (Neither)
   - D (Drop)

3. Enter the selected type of role in the *Role* column, next to the parameter you are defining. The options are:
   - R (Restrict)
   - M (Map)
   - V (Vary length)
   - P (Position)

Parameter Usage Restrictions

When the function option *Duplicate Parameters* is set as *Y*, a parameter field can appear on a function's parameter interface. This can occur anytime from none (zero) to numerous times with any parameter usage.

However, when the function option *Duplicate Parameters* is set as *N*, the following situation applies:

- For a given function, each parameter field must be unique. In other words, a field can only appear once regardless of usage (I,O,B, or N). The exception to this situation is when a field can be defined separately once for input and once for output, so it can appear two times.

Note: An attempted violation of this restriction and exception causes the error message *Y2V0214–Parameter is duplicate* to be sent.

Parameter Usage Matrix

When you have multiple items, certain combinations valid and certain combinations are invalid. For invalid combinations, processing does not allow arrays to be passed in the PAR context, where any subfield has usages on the Calling and Called program when the Compatibility is *Invalid, as shown in the following usage compatibility matrix:

Calling	Called	Compatible
N	N	Valid
N	I	Valid
N	O	Valid
N	B	Valid
I	N	Valid
I	I	Valid
I	O	*Invalid
I	B	*Invalid
O	N	Valid

Calling	Called	Compatible
O	I	*Invalid
O	O	Valid
O	B	*Invalid
B	N	Valid
B	I	Valid
B	O	Valid
B	B	Valid

Defining Parameters While in the Action Diagram

While in the action diagram, you can change the parameters of the function you are editing using the following instructions. This is useful when adding additional parameters while in the action diagram.

To define Parameters in the Action Diagram

1. At the Edit Database Relations panel, type **F** to view the selected function. The Edit Function panel appears.
2. View the Action Diagram. Type **F** next to the selected function. The Edit Action Diagram panel appears.

   In the Action Diagram, press **F9**. The Edit Function Parameters panel appears.
3. Define the parameter. Use the instructions in the two previous topics to define the parameter and specify the role and usage.
4. Press Enter to accept the changes.

   Press **F3** to exit and return to the action diagram. The action diagram redisplays with your changes.

For more information on action diagrams, see the chapter Modifying Action Diagrams (see page 431).

Specifying Parameters for Messages

A parameter can be used within the text portion of a message. During execution, the parameter's value displays.

To specify a parameter for a message function

1. Use the previous instructions to get to the Edit Message Functions panel.
2. Type **P** next to the selected message function. The Edit Function Parameters panel appears.
3. Define the parameter.

   Note: When the data type of a parameter allows value mapping, such as all date and time fields, the parameter is typically converted to its external format before the message is sent. However, due to limitations within i OS, the parameter data for the TS# data type is passed in its internal format, namely, YYYY-MM-DD-HH.MM.SS.NNNNNN.

A parameter can be defined for a message function to allow substitution of the parameter's value into the text portion of the message identifier.

For example, to insert a field's value in an error message when the credit limit is exceeded for a customer, enter the following:

```
Credit limit exceeded for &1.
```

The parameter value &1 is inserted into the message text at execution time. You must then define (&1) as an input parameter value to the message function. If this is an error message, it also causes the field associated with the parameter (&1) to display using the error condition display attribute for the field. By default, this is reverse image.

Using Arrays as Parameters

You can create elements in an array that are similar to parameter definitions, and you can pass certain parameters as an array. For example, multiple instances of data can be passed within the parameter.

By passing a parameter as an array, multiple instances of data can be passed in or out in a single call to a function. For example, if a customer record structure is defined in the Customer array on the *Arrays file, you can use that array to define a parameter to an EXCEXTFUN or EXCUSRPGM being passed as RCD (ARRAY). Anywhere from a few to thousands of customer records can be passed in that one parameter in one single function call.

Arrays are defined over the *Arrays file and can be defined to contain any subset of the fields in the model. Arrays can also be specified as parameters to any function. Using arrays allows you to define any subset of fields as parameters to any function. Do this by creating an array definition with the appropriate field and specifying it as a parameter entry. This process is similar to using structure files for parameter lists but unlike structure files, it is under the control of *PGMR.

Notes:

- With this process, you use the array to supply a parameter definition and not the data.
- When generating functions using SQL, an array used to define the parameters for CHGOBJ or CRTOBJ must have the fields defined in the same order as the update access path.

You can pass an array as a parameter using one of two methods:

- **Multiple-instance array parameter:** Describes when a parameter is passed as an array (when the *Pass as Array* flag is set to 'Y'). The parameter contains multiple instances of data, where each instance contains all the fields which are individually specified as parameters using the parameter details display.
- **Single-instance array parameter:** Describes when a parameter defined using an array is not passed as an array (when the *Pass as Array* flag is not available or is set to blank). The parameter contains all the fields which are individually specified as parameters using the parameter details display.

For more information about arrays, see the chapter "Defining Arrays" in the *Building Access Paths Guide*.

Multiple-Instance Restrictions

When passing a multiple-instance array the following restrictions apply:

- Only EXCEXTFUN and EXCUSRPGM allow parameters to be passed as a multiple-instance array.
- Parameters can only be passed as a multiple-instance array when the parameter structure is defined using an array based over the *Arrays file.
- Parameters can only be passed as a multiple-instance array when they are being passed as RCD or KEY.
- No fields can be dropped on a parameter being passed as multiple-instance array.
- Do not allow a multiple-instance array parameter in a function call, either in ARR nor PAR context, except when calling an EXCEXTFUN or EXCUSRPGM, that has a multiple-instance array parameter. Additionally, the call must be from the top level action diagram of an EXCEXTFUN function.
- The Submit job (SBMJOB) feature and Y2CALL command do not support function calls that contain multiple-instance array parameters

When working with two functions, function A and function B, for example, you can model in the action diagram of function A a call to function B, where B has a parameter interface passed as an array. In this case these additional restrictions apply:

- Function A must be of type EXCEXTFUN, and function B must be of type EXCEXTFUN or EXCUSRPGM.
- The parameter context must be PAR, ARR, or PR1 through PR9 for Duplicate Parameters, and the array name must exactly match on the parameter definition of A and B.
- If a parameter is passed as a multiple-instance array on A it must be passed as an array on B.
- The multiple-instance array parameter must be passed as RCD on both A and B, or KEY on both A and B.
- Although the usages of the subfields on a parameter passed as an array can be mixed, the usages must be compatible, such that the calling function can call the called function.

  For more information, see the section Parameter Usage matrix (see page 282).

Chapter 9: Modifying Device Designs

The purpose of this chapter is to introduce you to CA 2E device designs, to explain default device designs, conventions and styles, and to identify how to modify device designs for panels and reports.

This section contains the following topics:

Understanding Device Designs

A device design specifies the layout of fields and constants on panels or report designs that are associated with a function. There are two types of device designs:

- Panel designs, which specify the layout of fields and constants for interactive functions
- Report designs, which specify the layout of fields and constants for report functions

Both types of device designs are similar in overall structure and are modified with similar editors. However, some features apply to each specific device design.

If a CA 2E function has a device design associated with it, a default design is created by CA 2E when you create the function. You can then modify this design.

CA 2E animation provides a direct link between CA 2E and CA 2E Toolkit prototyping functions. This includes converting CA 2E device designs to Toolkit panel designs, full access to all Toolkit editing and simulation functions, and the ability to return directly to your CA 2E model. In addition, existing Toolkit navigation, narrative, and data are preserved when you download a new version of a panel design.

The device design for a function on the iSeries is implemented as a single i OS device file, a display file for panel designs or a print file for report designs.

A device design specifies the following:

- Which fields are present on the panel or report
- The position of fields and constants on the panel or report
- The circumstances under which particular fields are displayed
- Whether the field is input capable or protected (interactive panels only)
- Whether the field is optional or required (interactive panels only)
- The display attributes and editing of fields on the panel or report

Basic Properties of Device Designs

The three basic properties of device designs are design standard, formats, and fields.

Design Standard

The overall layout of the design is determined by the standard header/footer selected and the function type. Each device function is associated with a standard header/footer function of type Define Screen Format (DFNSCRFMT) or Define Report Format (DFNRPTFMT). These functions cannot stand-alone. You can only generate and compile the functions to which they are attached. These header/footer functions should only be defined on the CA 2E shipped file *Standard Header/Footer.

The standard header/footer functions specify a standard layout for device headers and footers. You can create your own version of these functions that you attach to the *Standard header/footer file using the Edit Functions panel and associate them with your device functions using the Edit Function Options panel for each device function. To create a new header/footer, you can copy and modify an existing one or add a new one.

When you define a new device function, a header/footer is automatically assigned to it according to defaults specified by your model values or settings on the header/footer functions.

For more information on standard header/footers, see the Standard Header/Footer topic later in this topic.

Presentation Convention for CA 2E Device Designs

The appearance of fields on the device designs as illustrated in this chapter is denoted by the following symbols.

Symbol	Definition
I	Input capable alphanumeric field
O	Output only alphanumeric field
B	Update alphanumeric field
3	Input capable numeric field
6	Output only numeric field
9	Update numeric field

For example, on a panel design:

```
                    Enter Orders
            Customer . . IIII 000000000000

O    Code      Name           Quantity    Price
I    BBBBBB    00000000000    99999.99    666.66
I    BBBBBB    00000000000    99999.99    666.66
I    BBBBBB    00000000000    99999.99    666.66
```

And on a report design:

```
                 Print Orders

Customer . :  0000000000000000

    Code     Name          Quantity    Price
    000000   00000000000   66666.66    666.66
    000000   00000000000   66666.66    666.66
    000000   00000000000   66666.66    666.66
                              Total:  6666.66
```

Note: The symbols used in these examples are for this module only and do not represent the actual method used in CA 2E.

Default Device Design

CA 2E provides default device designs based on the function options, standard header/footer, function type, and access path. The first time you enter the device design, CA 2E defaults the design for you according to the function type, access path, model values, and function options.

Device Design Formats

A function's device design is created from a number of device design formats, each of which specifies part of the device design. Each format is created from the fields of the based-on access path and their associated text. Each format has:

- **Device file format details**—For each format, you specify format level information, for instance, descriptive text and information as to how to position the format relative to the other formats on the design. In the case of report design formats, overflow criteria are specified. CA 2E supplies appropriate defaults for this information.
- **Device file format entries**—The format entries constitute a list of all the fields that appear in that format on the display panel along with information on how these fields are to appear. Fields are positioned by default on the display in the order in which their entries appear on the access path. You can change the order of entries, the positioning, and remove or add entries.

Device Design Fields

CA 2E gets fields for a default device design from the following three sources:

- Header/Footer associated with a device function
- Access Path to which the function attaches
- Function parameters

Header/Footer Associated with a Device Function

The header/footer associated with a device function can contain a number of different fields including panel title, job, and user. These fields are on the CA 2E shipped file *Standard header/footer. You can add relations to this file if you need additional fields but you are responsible for filling the fields with data.

Access Path to Which the Function Attaches

All fields in the access path are included on the device design by default when the function is first created. If fields are added to the access path once the function is created, they are available as hidden fields in the appropriate format and can be set to input or output. They can then be moved to the appropriate place on the device design.

Note: Virtual fields are output only.

Function Parameters

Function parameter fields with a role of *mapped* are included on display device designs.

If these parameter fields correspond to access path entries, the parameters are mapped into the existing entry. If they do not correspond, a new entry is added to the panel.

You can also add the following types of fields to the individual device designs:

- Function fields
- Constants

Panel Design Elements

The panel design usually consists of certain basic elements. These elements are based on whether you use a multiple record function (EDTFIL, DSPFIL, or SELRCD), a single Record Function (EDTRCD 1,2,3, DSPRCD 1,2,3, or PMTRCD), or a transaction function (EDTTRN or DSPTRN). The elements are listed below:

- Multiple Record Function
  - Standard Header
  - Subfile Control
  - Subfile Records
  - Standard Footer
- Single Record Function
  - Standard Header
  - Key Screen
  - Detail Screen
  - Standard Footer
- Transaction Function
  - Subfile Standard Header
  - Control
  - Subfile Records
  - Standard Footer

The following is a table of CA 2E display function formats.

Function Type	HDR	FTR	SFLCTL	SFLRCD	DTL
PMTRCD	Y	Y	–	–	Y
DSPRCD	Y	Y	–	–	Y
EDTRCD	Y	Y	–	–	Y
DSPFIL	Y	Y	Y	Y	–
EDTFIL	Y	Y	Y	Y	–
SELRCD	Y	Y	Y	Y	–
DSPTRN	Y	Y	Y	Y	–
EDTTRN	Y	Y	Y	Y	–

Panel Body Fields

The fields that appear by default on the panel body are derived from the access path on which the device function is based and from the function parameters. You can add further fields.

When laying out default panels, CA 2E treats different types of fields in different ways:

1. Parameters. The role of the parameters has a significant effect on how they are used on the default device design.

   For more information about parameters, see the chapter, "Modifying Function Parameters."

2. Key fields in the based-on access path. For most of the standard function types there are constraints as to how key fields can be used. In some cases (for instance, the EDTRCD key panel), they must be input capable; in others (such as on DSPFIL subfile records), they must be protected. In DSPFIL, EDTFIL, and SELRCD function types, positioner fields are also provided on the subfile control record for each key field.

3. Non-key fields in the based-on access path. There are fewer restrictions as to how non-key fields can be used. In the DSPFIL and SELRCD function types, selector fields are also provided on the subfile control record for each non-key field.

General Rules for Panel Layout

The following information identifies features common to all display styles.

The first two lines of the panel designs contain a title and a status information defined by an associated DFNSCRFMT function. When you create a new model, four layouts are provided automatically:

- CUA Entry standard
- CUA Text Subset—Action Bars
- CUA Text Subset—Windows

Lines twenty-three or twenty-two contain text explaining the meaning of any function keys. The format of this text depends on the value of the YSAAFMT model value, either CUA Entry or CUA Text.

Messages appear on line twenty-four. Message clearing and resending is controlled by the function options.

For more information about function options, see the chapter, "Modifying Function Options."

Panel Layout Subfiles

For subfile record fields, Column Heading text is used; for other fields, the Before text is used.

A subfile selector field, *SFLSEL, is added to the beginning of the subfile record for device designs that include subfiles, providing the appropriate function option is specified.

Text that explains the meaning of the selection values is provided if appropriate. The positioning of this text depends on the value of the Enable Selection Prompt Text function option on the associated standard header function. This can follow the CUA convention of text placed above the subfile text. If the Selection Prompt text is chosen to be above the subfile, extra fields, such as *PMT, *SELTXT, are added automatically to the display.

Panel Layout Field Usage

Any fields that are restrictor parameters are given an output-only usage.

For more information on restrictor parameters, see the chapter, "Modifying Function Parameters."

If the function is an Edit function (EDTRCD, EDTFIL, EDTTRN), the non-key fields from the access path on which the function is based are input capable on the panel unless they are virtual fields or specifically protected. The key fields from the access path are only input capable when records are being added.

If the function is a Display function (DSPRCD, DSPFIL, SELRCD, DSPSTRN), the fields from the access path are output only.

Any fields used to control the positioning of a subfile display appear on the subfile control record at the top of the display. These fields are input fields.

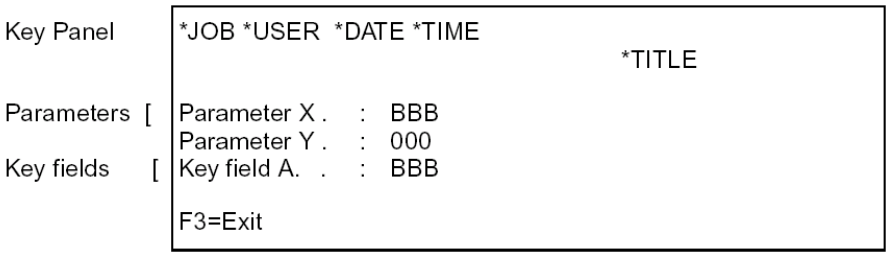
Virtual fields are added to the device design immediately after the real fields with which they are associated. Virtual fields are always output only.

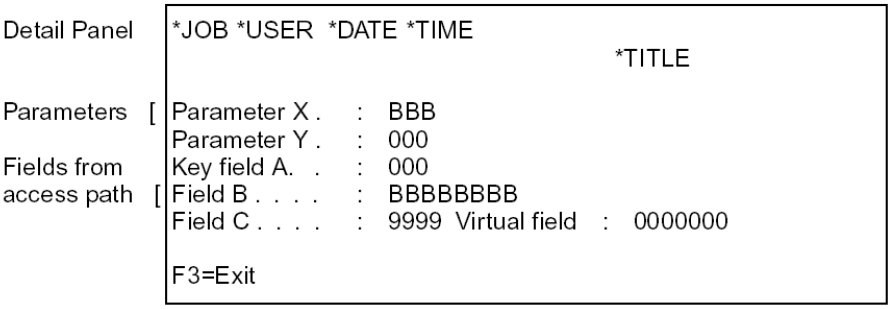
If a field is added to an access path, it is added to the panel design as a hidden field.

Default Layout of a Single-Record Panel Design

For single-record style panels (EDTRCD, DSPRCD, PMTRCD), CA 2E lays out the fields on the panel design as follows:

- Key fields from the based-on access path are placed, one field per line, on both key and detail panels.
- Non-Key fields from the based-on access path are placed, one field per line, on detail panel designs.
- When the YCUAEXT model value is set to *DEFAULT, virtual fields are placed on the same line as the real field with which they are associated. When the YCUAEXT model value is set to *CUATEXT, the virtual fields indent three spaces on the following line.
- Map function parameter fields that cannot be mapped to any existing field are placed on the display before the other fields.



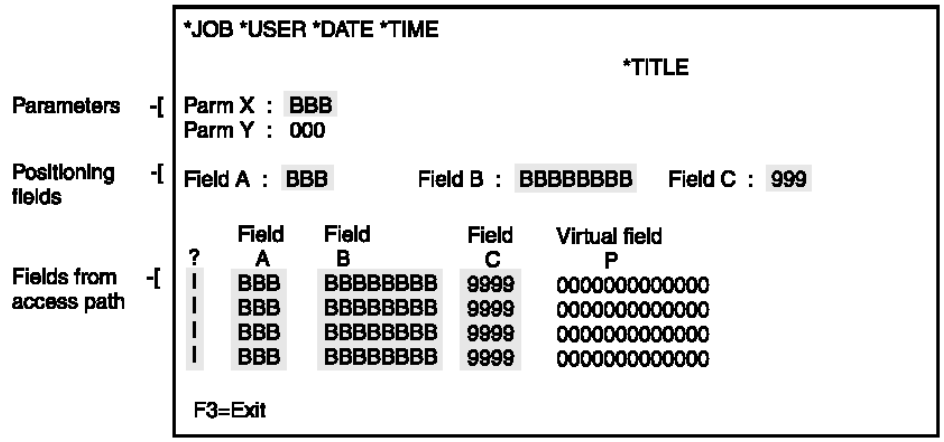


If there are more fields than will fit on a panel, the design may run over onto additional pages.

Default Layout of a Multiple-Record Panel Design

For multiple-record style panels (EDTFIL, DSPFIL, SELRCD), CA 2E lays the fields out on the device display as follows:

- Non-restrictor fields from the format of the based-on access path are placed on each subfile record, one field after another on the same line. Any fields, which are defined as restrictor parameters are omitted from the subfile record and are, instead, placed on the subfile control record.
- Key fields from the access path that are also restrictors are placed on the subfile control record.
- For each key field in the based-on access path, a positioner field is placed on the subfile control record. This field can be used to position the loading of the subfile to start at a particular database record.
- In DSPFIL and SELRCD function types, for each non-key field in the based-on access path, a selector field is placed on the subfile control record. The nature of the selection can be specified using the Edit Screen Entry Details panel.
- Map function parameter fields that cannot be mapped to an existing field are placed on the panel, one field per line, before the other fields.

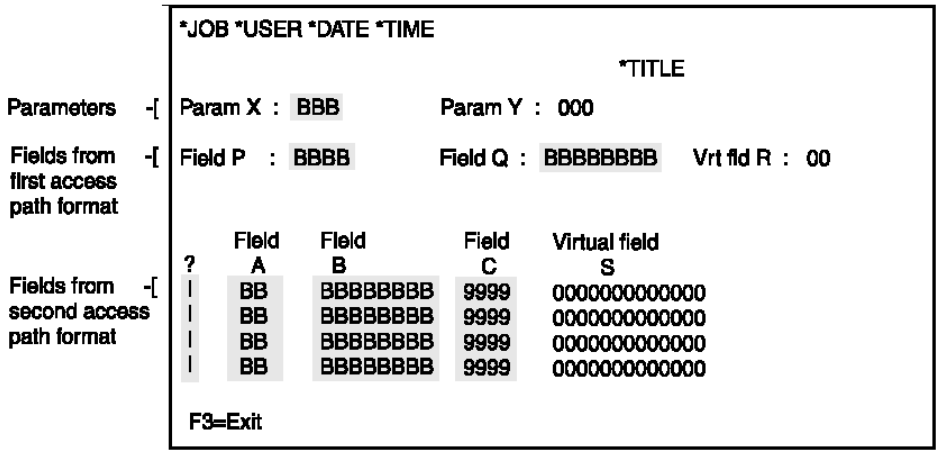


If there are more fields than will fit on a panel, the design can extend past position 80 to 132, provided you have the appropriate terminals. If you require a 132 display, you need to set the function option for 132 on your standard header/footer.

Default Layout of a Single- and Multiple-Record Panel Design

For single- and multiple-record style panels (EDTTRN, DSPTRN) that must attach to an SPN access path possessing two formats, CA 2E lays out the fields on the device design as follows:

- The common major key or keys of the two formats of the based-on access path are placed on the subfile control record
- Fields from the first format of the access path are placed on the subfile control record one after another, on the same line
- All fields from the second format of the access path, apart from the shared key, are placed on each subfile record, one field after another on the same line.
- Parameters that cannot be mapped to existing fields are placed on the subfile control record, one after another on the same line.



National Language Design Considerations

You should consider the following when creating device designs and layouts for applications that is used to support National Languages.

- Allow 25% to 50% additional space in your device design for National Language Support (NLS.
- Avoid the use of multi-column headings
- Do not use abbreviations or symbols
- Follow CUA standards
- Do not clutter panels; if necessary, use additional panels

The following design considerations can be affected when you develop your application with NLS:

- DBCS considerations
- Bi-directional considerations
- Presentation functions
- Help Text

Note: When designing applications for National Languages, be sure to set the model value YPMTGEN to *OFF until you are ready to generate your final production model. This provides you with a performance benefit during function design. Once you are ready to generate your final model, you can set it to *MSGID.

For more information on National Language Support and on changing or creating applications in other languages, see Generating and Implementing Applications, in the chapter, "National Language Support."

Device Design Conventions and Styles

These are the display conventions that are used to create default panel designs for functions are:

- CUA Text
- CUA Entry

The display conventions affect various features of the panel design including:

- Position of the panel title and fields used on the panel header
- Default function keys (such as F3 for Exit)
- Position and style of the function key and subfile selector text
- Default display attributes of the fields in the screen body
- Style of dot leaders used to connect fields with their field text
- Use of windows and action bars

You can choose either CUA Entry or CUA Text using the YSAAFMT model value. CA 2E generates the:

- CUA Entry standard when you set this value to *CUAENTRY
- CUA Text subset when you set this value to *CUATEXT, and System/38 standard when you set this value to *S38

CUA Text

Selecting CUA Text as a default enables you to generate applications with windows and action bars. Action bars use their associated pull-down menus. The CUA Text standards provide panels (which have action bars) and windows (which do not have action bars) for the generated application by default.

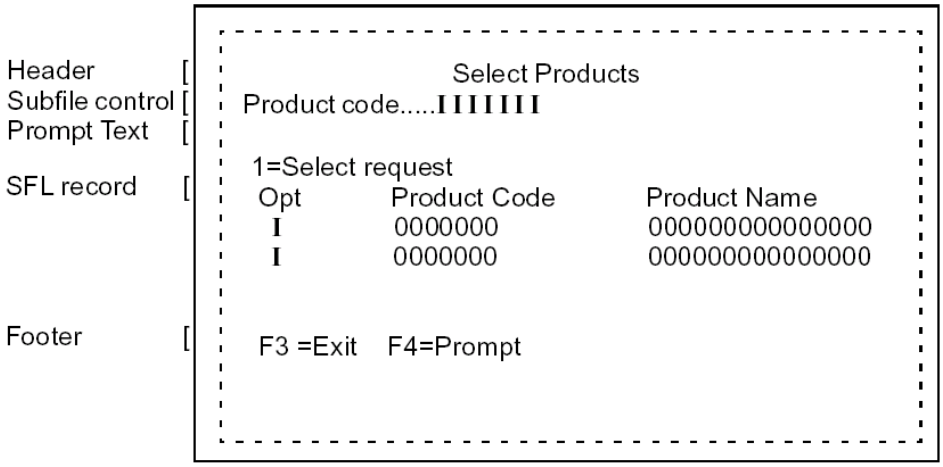
Note: Another term for action bar is menu bar.

Windows

A window is an area of a panel with visible boundaries in which information appears. Windows do not have an action bar. Windows can overlap on the panel, one panel superimposed on another. Only the topmost window is active.

When you set the model value YSAAFMT to *CUATEXT, newly created device functions default to action bars and windows for generated applications. Only Select Record (SELRCD) functions default to a standard window header/footer but you can make any other function a window by selecting a window header/footer for it from the function options.

CUA Text Window

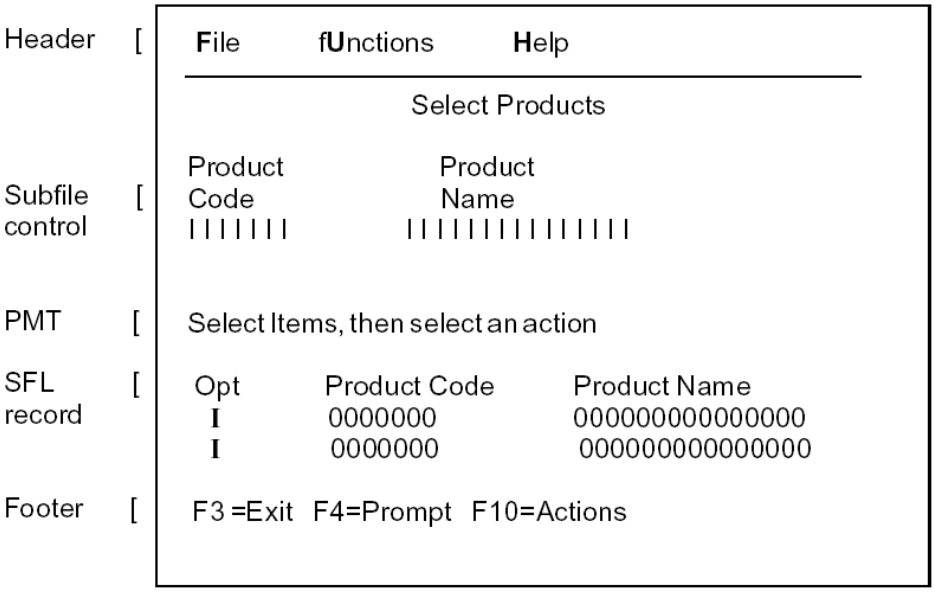


Action Bar

An action bar appears at the top of a panel and provides a set of choices and actions across the top of the panel. The choices allow end users access to the actions available from the panel. Depending on the function, CA 2E provides logical defaults for action bar choices, action bar mnemonics, pull-downs, associated descriptive text, and pull-down accelerator keys.

When you set the YSAAFMT model value to *CUATEXT, all newly created interactive device functions default to action bars, with the exception of SELRCD. SELRCD functions default to the window header/footer.

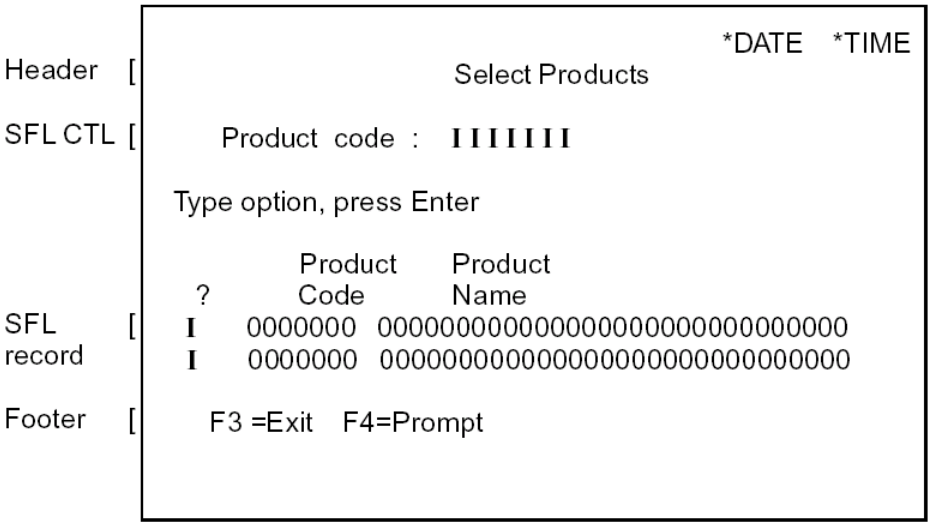
CUA Text Action Bar



CUA Entry

CUA Entry is a standard header/footer device format. Selecting a CUA Entry header/footer gives you a panel that is designed to comply with IBM's CUA '89 Entry Model recommendations. The default header/footer for CUA Entry is *Std Screen Headings (CUA).

CUA

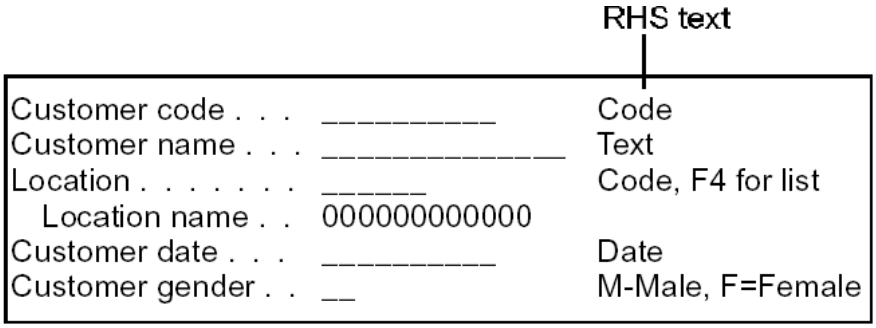


System 38

CUA Device Design Extensions

CUA device design extensions consist of the following additional CUA design features which are controlled by setting the model value YCUAEXT to *C89EXT:

- **Automatic defaulting and alignment of right-hand side text**—For input capable fields in the generated application, CA 2E automatically supplies and aligns right-hand side (RHS) explanatory text. RHS text does not appear on panel fields. For the application user, this text indicates the allowed values. The text appears aligned and to the right of the field, as shown below.



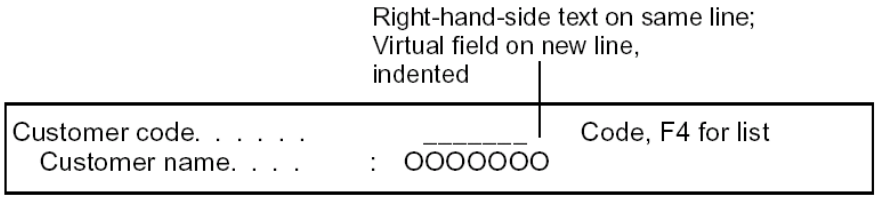
- **Padding of field constant or literal trailers (dot leaders) for left-hand side text**—CA 2E automatically adds padding characters to the end of field left-hand side (LHS) text labels for panel appearance and to facilitate translation into other languages. The amount of padding depends on the label length, as described below. The resulting length is rounded to an even number.

Field length	Percentage
0-12	100%
12-29	50%
30-59	30%
60+	0%

- **Display of prompt instructions for device functions**—With *C89EXT, prompt instructions appear for all device functions.

Prompt Instructions	Function Type
Type options, press Enter.	EDTFIL, EDTTRN, DSPFIL, DSPTRN, SELRCD
Type choices, press Enter.	EDTRCDn key, DSPRCD key, PMTRCD
Type changes, press Enter.	EDTRCDn detail
Press Enter to continue	DSPRCDn detail

- **Indenting of virtual fields**—For the initial layouts of Display Transaction (DSPTRN) and Edit Transaction (EDTTRN) header formats, and for Prompt Record (PMTRCD), Display Record (DSPRCD), and Edit Record (EDTRCD) detail formats, CA 2E makes space for right-hand side text by moving the virtual fields associated with a foreign key to the next line and indenting them three spaces, as illustrated below.



Rightmost Text

CA 2E automatically supplies and aligns RHS explanatory text for input capable fields in the generated application. RHS text does not appear for display fields. For the application user, this text indicates the data field type and the type of data that should be entered, such as text, number, and allowed values. The text appears aligned and to the right of the field. RHS text appears when the YCUAEXT model value is set to *C89EXT.

Panel Defaults for Rightmost Text

The defaults for RHS text are based on field attribute type and a number of CUA design considerations such as role, entry type, field validation, and usage. The defaults are automatic for new fields.

If the role of the field is to position information such as that of some fields on Display File (DSPFIL), Edit File (EDTFIL), and SELRCD functions, a special value of Starting Characters is used for the RHS text. If the field is a foreign key field, then F4 for list is appended when YCUAPMT is set to *YES.

If a check condition is defined for the field, the RHS text is built from the allowed values according to the condition type, the value length, and the number of values allowed.

For example, for a Status (STS) field with a List (LST) check condition with the two Value (VAL) conditions Male and Female, the RHS text default is M = Male, F = Female.

The panel RHS flag defaults for a field according to the field‘s usage, as described in the following table:

Type	RHS Text Style	Substitution Values
CMP	&1 &2 Example: *GT 5	&1 = *Relational operator &2 = Value
RNG	&1 - &2 Example: 1001-4005	&1 = From value &2 = To value
LST (1)	&1 = &2, &3 = &4 Example: M = Male F = Female	&1 = Value1 &2 = Condition1 &3 = Value2 &4 = Condition2
LST (2)	Value, F4 for list	-

Type	RHS Text Style	Substitution Values
LST (3)	&1, &2, &3, &4... Example: *ADD, *CHG, *DLT	&1 = Value 1, &2 = Value 2, &3 = Value 3, &4 = Value 4,

RHS text aligns two spaces after the longest field. CA 2E aligns RHS text when you create a new panel and when you request field realignment at Edit Device Design. A field on the Edit Device Design panel allows you to override the number of default spaces between a field and the right-hand side text.

The CA 2E field defaults for RHS text are based on field attribute types, as described below. The defaulting is automatic for new fields.

CA 2E Field Attribute		Right-Hand side text Default
CDE	Alphanumeric code value	Code
DT#	ISO Date	Date
DTE	Date in system date format	Date
NBR	Pure numeric value	Number
PCT	Percentage or market index	Percent
NAR	Narrative text	Text
QTY	Quantity	Quantity
STS	Status	Value
TM#	ISO Time	Time
TME	Time in HHMMSS format	HH:MM:SS
TS#	ISO Timestamp	Timestamp
TXT	Object text	Text
VAL	Monetary value	Monetary value
VNM	Valid System name	Name
PRC	Price or tariff	Price
IGC	Ideographic text	IGC Text

Standard Headers/Footers

CA 2E provides a file containing standard header/footer fields to which any functions defining headers and footers for use by device functions can be attached. CA 2E ships five default functions with this file. Four of them are Define Screen Format (DFNSCRFMT) functions. The other default header/footer is a Define Report Format (DFNRPTFMT) function, which defines report design headers. These default functions are:

- *Standard Report Heading
- *Standard Screen Heading
- *Std CUA Action Bar
- *Std CUA Window
- *Std Screen Heading (CUA)

You can modify these shipped versions as well as add your own DFNSCRFMT and DFNRPTFMT functions for use in specific function panel designs.

Note: The default date field on the standard header/footer uses the date format as defined in the job description of the person executing the application. The date on the header does not use the CA 2E model values YDATFMT and YDATGEN to determine the run-time format. The date format for this field can be controlled by the individual job description or the i OS QDATFMT system value.

Function Keys

CA 2E identifies a number of standard function key definitions, for example Exit, Rollup, and Delete. The standard functions refer to these definitions rather than to any particular function key. A function key is then assigned to each meaning. This makes it possible to change the user interface of an application simply by reassigning the function keys and regenerating the functions.

When you create a new model with the command Create Model Library (YCRTMDLLIB), the initial values for assigning the function keys are controlled by the DSNSTD parameter.

You can also specify alternative values for standard function key meanings. For example, you could specify F7 as the Exit function key.

The following standard function key meanings are used in the default device designs.

Meaning	ISeries Default
*Help	F01/HELP
Prompt	F04

Meaning	ISeries Default
Reset	F05
*Change mode request	F09
*Change mode to Add	F09
*Change mode to Change	F09
*Delete request	F11
*Cancel	F12
*Exit	F03
*Exit request	F03
*Key panel request/*Cancel	F12
*IGC support	F18
Change RDB	F22
*Previous page request	F07/ROLLDOWN
*Next page request	F08/ROLLUP

Note: For CUA Text, you can use F10 to activate the action bar.

Additional function keys can be specified using the Action Diagram Editor. For functions with an action bar, the command text defaults from the action bar accelerators.

For more information on action diagrams, see the chapter, "Modifying Action Diagrams."

IGC Support Function Key

The Ideographic Character (IGC) support condition assigns a function key to invoke i OS ideographic support using the DDS IGCCNV keyword. Note that this information is optional if your keyboard has an IGC mode key.

Note: Code is only generated for the IGC support function key if the model value (YIGCCNV) is set to 1.

The following table shows the default function keys by function type.

Function Type ISeries	*EXIT (F01) (F03)	*PREV (F02) (F12)	*ADD (F09) (F09)	*CHG (F09) (F09)	*DLT (F11) (F11)	HOME, ENTER, HELP	ROLLUP, ROLLDOWN
PMTRCD	Y	-	-		-	Y	-

Function Type ISeries	*EXIT (F01) (F03)	*PREV (F02) (F12)	*ADD (F09) (F09)	*CHG (F09) (F09)	*DLT (F11) (F11)	HOME, ENTER, HELP	ROLLUP, ROLLDOWN
DSPRCD	Y	-	-	-	-	Y	-
DSPRCD2	Y	Y	-	-	-	Y	Y
DSPRCD3	Y	Y	-	-	-	Y	Y
EDTRCD	Y	-	Y	Y	Y	Y	-
EDTRCD2	Y	Y	Y	Y	Y	Y	Y
EDTRCD3	Y	Y	Y	Y	Y	Y	Y
SELRCD	Y	-	-	-	-	Y	Y
DSPFIL	Y	-	-	-	-	Y	Y
EDTFIL	Y	-	Y	Y	-	Y	Y
DSPTRN	Y	-	-	-	-	Y	Y
EDTTRN	Y	-	Y	Y	Y	Y	Y

Function Key Explanations

Each panel design includes one or two lines of explanatory text for the function keys on the footer format. The text is built from the function key conditions referenced in the action diagram of the function (that is, references to conditions attached to the *CMD key field). Text for the HELP, HOME, and ROLLUP keys is omitted.

The number of lines of text (one or two), and the positioning of the function key explanations can be changed by altering the device design of the Standard header function associated with the device function.

The format of the text depends on the value of the model value YSAAFMT. It can follow the CUA (F3=Exit) conventions.

The text of the explanations can be changed for the function using the Device Design Editor. A default set of explanations is provided for the default function keys for each function type.

Specifying Function Keys

Function keys are defined in CA 2E as field conditions attached to the *CMD key field. All the allowed values are predefined in the shipped system. Refer to the section on the CA 2E environment to see how command key values are assigned.

For more information:

- On CTL context, and an example of the use of function keys, see Understanding Contexts in the chapter "Modifying Action Diagrams."
- On assigning function key values, see Changing the Number of Function Key Text Lines later in this chapter.

Subfile Selector Values

The same meaning is given to each subfile selector value across all function types. The options that are actually enabled depend on the function type. The following standard meanings are used in the default device designs:

*SFLSEL Condition	Meaning	CUA Entry Shipped Values	CUA Text Shipped Values
*Delete#1	Delete	4	-
*Delete#2		4	-
*Zoom#1	Show details for this item	5	-
*Zoom#2		5	-
*Select#1	Select this item	1	-
*Select#2		1	-
*Selection char value	Select item(s) for action	-	/
*Selection char value 2		-	/

Note: For CUA Text, delete and zoom are on the action bar.

Additional subfile selector values can be specified using the Action Diagram Editor.

Subfile selector values are specified as field conditions attached to the *SFLSEL (subfile selector) field. For functions with an Action Bar, the subfile selection text defaults from the Action Bar accelerators. The standard values are present in the shipped system.

Note: The length of the *SFLSEL field can be either one or two characters; it is shipped with a length of one. Any developer can override the model-wide length for a particular function on the Edit Screen Entry Details panel. A designer (*DSNR) can change the model-wide length of the *SFLSEL field using the Edit Field Details panel.

For more information:

- On the RCD context and the use of subfile selections, see Understanding Contexts in the chapter "Modifying Action Diagrams."
- On how to change the values assigned to subfile selector values, see Subfile Selector Value Explanatory Text later in this chapter.

  The following table shows the default selection options by function type.

Function Type	*SELECT	*DELETE
SELRCD	Y	–
DSPFIL	–	–
EDTFIL	–	Y
DSPTRN	–	–
EDTTRN	–	Y

Panel Design Explanatory Text

Panel designs can include two sorts of explanatory text:

- Explanations of the standard function key meanings
- Explanations of the standard subfile selector value meanings, such as 4-Delete on CUA Entry; Delete is an Action Bar choice on CUA Text

An initial version of this text is built automatically for each device design from the action diagram of the device function. You can then modify it.

The way the explanation text strings are built and the positions in which they are placed on panel designs depend on the interface design standards that you use. A number of variations are possible, controlled by the factors described below.

Positioning of the Explanatory Text

CA 2E lets you position explanatory text for function keys and subfile selector values.

Function Key Explanatory Text

Function key explanatory text is always placed in the position specified for the *CMDTXT fields on the standard header function (DFNSCRFMT) associated with the function whose device design you are editing. You can change the current standard header function using the function options display. One or two lines of text can be specified. If two lines of text are allowed but only one is needed, the text is placed in the lower of the two lines.

You can control whether the explanatory text appears by changing the usage of the *CMDTXT1 and *CMDTXT2 fields on the associated standard header function‘s device design.

Subfile Selector Value Explanatory Text

Subfile selector value explanatory text is only built if there are subfile selector values for the function.

You have a choice of two different positions in which to place any subfile selection explanatory text: you can either place it as part of the function key explanation text (normally at the bottom of the display), or you can place it as a separate line on the subfile control record (CUA standard).

The position at which subfile selector value explanatory text is placed depends on whether you specify that the *SELTXT fields on the subfile control field are to be displayed.

- If the *SELTXT field or fields (up to two lines are permitted) are visible, the explanation text appears at the position indicated by them on the subfile control format.
- If the *SELTXT fields are hidden, the selection value explanation text appears at the position indicated by the *CMDTXT field or fields as specified by the associated standard header function. If there is only one line of *CMDTXT, the subfile explanation text appears on the same line. If there are two lines of *CMDTXT, then subfile explanation text appears on the first line and the function key explanations on the second.

Whether the *SELTXT fields are available on panel designs is controlled by:

- Enable selection prompt text function option on the associated standard header function
- Usage (hidden H or output O) of the *SELTXT fields on the device display

This is shown by the following table.

DFNSCRFMT			Device Design		Resulting Position of Text
CMDTXT 1 usage	**CMDTXT 2 usage**	**Enable pmt txt**	**SELTXT1 Usage**	**SELTXT2 Usage**	
–	–	N	–	–	No text explanations
O	–		–	–	Selection and command text in *CMDTXT1 field
O	O	1	–	–	Selection in *CMDTXT1, Command text in *CMDTXT2

DFNSCRFMT			Device Design		Resulting Position of Text
CMDTXT 1 usage	CMDTXT 2 usage	Enable pmt txt	SELTXT1 Usage	SELTXT2 Usage	
O	–	1	O	–	Command text in *CMDTXT1 Selection in *SELTXT1
O	–	2	O	H	Command text in *CMDTXT1 Selection in *SELTXT1
O	O	2	O	O	Command text in *CMDTXT1&2 Selection in *SELTXT1&2

The selection text fields (*SELTXT1& *SELTXT2) can be preceded by a third field, the selection prompt field (*PMT) contains an explanation of how to use the explanation fields.

For example: Type: **option**, press Enter.

Form of the Explanatory Text

To build the text, CA 2E examines the action diagram to determine how the function keys and subfile selection values are used. It then uses the condition name associated with each function key condition or selection value condition found to create a text string according to the design standard specified by the model value YSAAFMT.

CUA Entry Format

If the YSAAFMT model value has the value *CUAENTRY, text has the form:

```
nn=Action nn=Action
Fn=Function Fn=Function
```

For example:

```
D=Delete
F3=Exit F9=Change mode
```

CUA Text Format

If the YSAAFMT model value has the value *CUATEXT, text has the form:

```
/ = Select(1)
Fn = Function Fn = Function
```

For example:

```
/ = Select
F3 = Exit F9 = Change mode
```

Specifying Panel Design Explanatory Text

For functions that can operate in more than one mode, there can be two versions of the explanatory text, one for each mode in which the function can operate. You can change or add to the explanatory text for the different modes using the Edit Screen Design Command Text panel that is available from the Edit Screen Design panel.

Changing the Number of Function Key Text Lines

You can specify alternative values for standard function key meanings. For functions with an action bar, the function key text defaults from the action bar accelerators.

Each panel design includes one or two lines of explanatory text for the function keys on the footer format. The text is built from the function key conditions referenced in the action diagram of the function, for example, references to conditions attached to the *CMD key field. Text for the HELP, HOME, ROLLUP, ROLLDOWN, and ENTER keys is not displayed on the panel design.

The number of lines of text (one or two) and the positioning of the function key explanations can be changed by altering the device design of the standard header function associated with the device design.

Function key explanatory text is always placed in the position specified for the *CMDTXT fields on the standard header function Design Screen Format (DFNSCRFMT) associated with the function whose device design you are editing. You can specify one or two lines of text. If you specify two lines and only one line is needed, the text is placed in the first line.

To change the number of function key text lines:

1. Go to the Function Options panel for the header/footer default.
2. Change the enable selection prompt text field (number of selection prompt text lines).
3. Change the usage of the *CMDTXT1 or *CMDTXT2 accordingly.

Note: You are changing the standard header/footer default. All footers associated with this default are affected.

Table of Panel Design Attributes

Screen Design Attribute	Initially Set By	Changed by	Override on Standard Header/ Footer	Override on Function
Default header/footer and Action Bars and Windows	DSNSTD parameter of YCRTMDLLIB	Default option on header/ footer or YCHGMDLVAL for YSAAFMT	N	Y
Position of selection text	Dependent on default Standard header/footer	Altering Standard header/footer	Y	N
Style of function key and selection text explanations	DSNSTD parameter of YCRTMDLLIB	YCHGMDLVAL for YSAAFMT	N	Y

Screen Design Attribute	Initially Set By	Changed by	Override on Standard Header/ Footer	Override on Function
Default function keys	DSNSTD parameter of YCRTMDLLIB	Change LST conditions for *CMDKEY in model or YCHGMDLVAL for YSAAFMT	N	N
Fixed display attributes	DSNSTD parameter of YCRTMDLLIB	YEDTDFTATR	N	Y
Dot leaders	DSNSTD parameter of YCRTMDLLIB	YCHGMDLVAL for YLHSFLL,YCUAEXT	N	N
Right-hand side text	YCUAEXT set to *CUA89	YCHGMDLVAL for YCUAEXT	N	N
Virtual Field Indenting	YCUAEXT set to *CUA89	YCHGMDLVAL for YCUAEXT	N	N
Prompt	YCUAPMT set to *MDL	YCHGMDLVAL for YCUAPMT	N	N

Editing Device Designs

The default device designs that are created when the functions are defined can be modified to suit specific needs. This topic provides you with information on how to make changes to the default device designs.

For more information on prototyping your CA 2E device design using Toolkit, see the *Implementation Guide* and the *Toolkit Concepts Guide*.

Editing the Device Design Layout

CA 2E lets you edit the layout of your device design to display the information that you need to see. Only 80 characters of the panel design can display at a time. This means there are times when all fields in a record cannot be displayed. There are function keys that allow you to shift the panel horizontally, moving left to right, to realign or to display the selected fields of your record that exceed the 80-character design layout.

Depending on where you are in CA 2E, you have more than one option for getting to your device design. Use one of the following sets of instructions.

Note: These instructions are only provided here, in the beginning of this chapter. Other instructions in this chapter assume that you are at the device design level.

From the Edit Database Relations Panel

1. View the list of functions. Type **F** next to the selected relation and press Enter.

   The Edit Function panel appears.

2. Type **S** next to the selected function and press Enter.

   The device design for the selected function appears.

From the Open Functions Panel

Type **S** next to the selected function and press Enter.

The device design for the selected function appears.

From the Edit Function Details Panel

Press F9.

The device design for the selected function appears.

From the Edit Model Object List Panel

Type **17** next to the selected function and press Enter.

The device design for the selected function appears.

Changing Fields

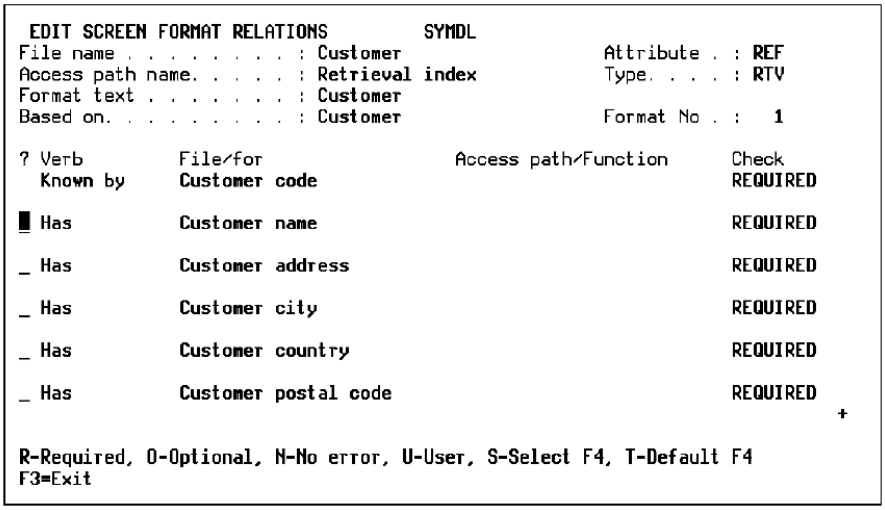
You can change the usage of a field on a device and conditionally set the display attributes.

To Change the panel's format relations

1. Select the field. At the device design, place the cursor on the selected field and press F7.

   Note: The selected field must be a field in the first subfile record excluding the subfile select field.

   The Edit Screen Format Relations panel appears.



Note: Use the Edit Report Format Relations panel for report designs.

2. Change the format relations. Select one of the following options to change the format relations and press Enter. An explanation of each option is provided.
   - O = optional
   - R = required
   - N = no error
   - U = user checking
   - S = select alternate prompt function
   - T = cancel alternate prompt function selection

For more information on the format relations options, see the Editing Device Design Formats later in this chapter.

CA 2E optional F4 prompt function assignment enables you to override the function assigned to the access path relationship at the access path and function levels. Using F4 prompt function assignment you can assign any external function other than Print File to the relation. The relation must be a file-to-file relationship.

To Assign the Override at the Function Level:

1. At the Edit Screen Format Relations panel, type S next to the selected relation and press Enter.

   The Edit Function panel appears.

2. Type **X** next to the selected function and press Enter.

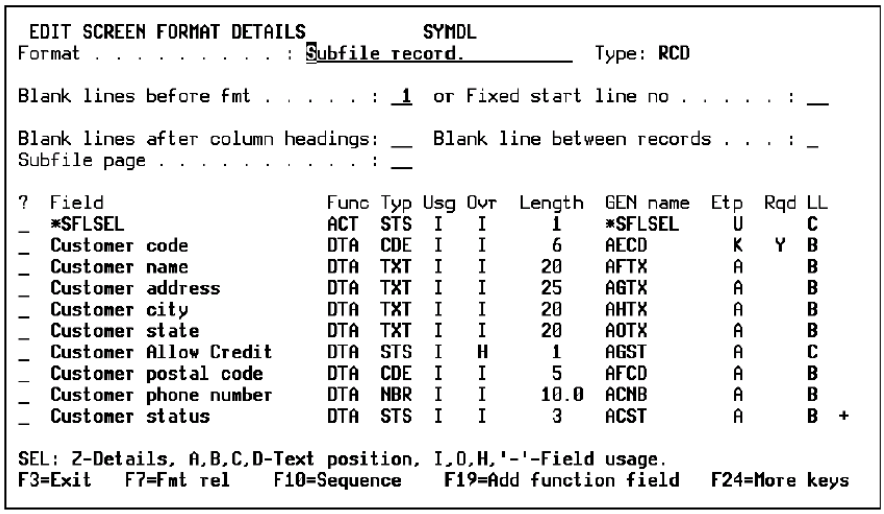
To Cancel the Override Selection type **T** next to the selected relation to turn off the selection override. CA 2E now overrides the default function assigned to the access path relationship.

For more information on F4 prompt function assignment, see SELRCD in the chapter "Defining Functions."

To Change the Field's Format Details

1. Select the field. At the device design, place the cursor on the selected field and press F5**.**

   The Edit Screen Format Details panel appears.



```
 EDIT SCREEN FORMAT DETAILS          SYMDL
Format . . . . . : Subfile record.          Type: RCD

Blank lines before fmt . . . . . : 1  or Fixed start line no . . . . . : __

Blank lines after column headings: __  Blank line between records . . . : _
Subfile page . . . . . : __

?  Field                    Func Typ Usg Ovr  Length  GEN name  Etp  Rqd LL
_  *SFLSEL                  ACT  STS  I   I       1    *SFLSEL    U       C
_  Customer code            DTA  CDE  I   I       6    AECD       K   Y   B
_  Customer name            DTA  TXT  I   I      20    AFTX       A       B
_  Customer address         DTA  TXT  I   I      25    AGTX       A       B
_  Customer city            DTA  TXT  I   I      20    AHTX       A       B
_  Customer state           DTA  TXT  I   I      20    AOTX       A       B
_  Customer Allow Credit    DTA  STS  I   H       1    AGST       A       C
_  Customer postal code     DTA  CDE  I   I       5    AFCD       A       B
_  Customer phone number    DTA  NBR  I   I    10.0    ACNB       A       B
_  Customer status          DTA  STS  I   I       3    ACST       A       B  +

SEL: Z-Details, A,B,C,D-Text position, I,O,H,'-'-Field usage.
F3=Exit   F7=Fmt rel    F10=Sequence    F19=Add function field   F24=More keys
```

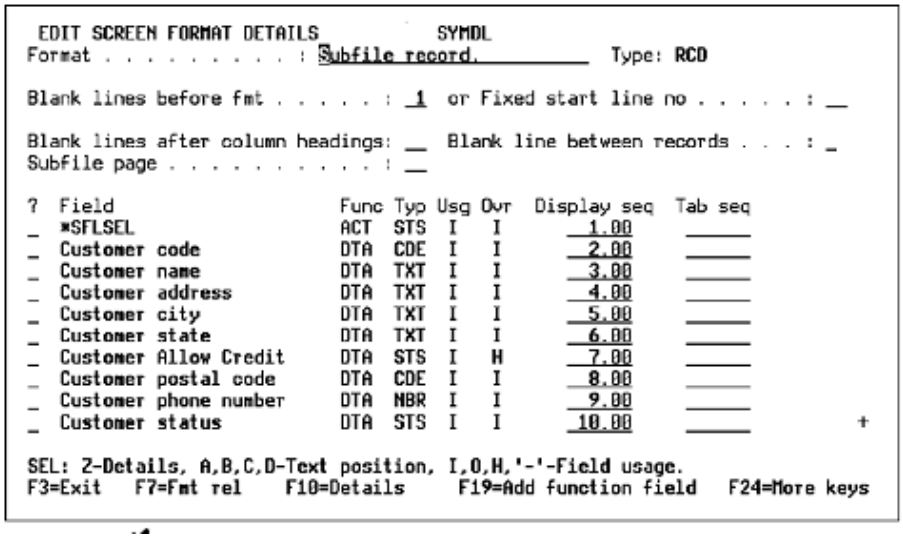
Or,

Place the cursor on the selected field and press Enter.

The Edit Report Entry Details panel or the Edit Screen Entry Details panel appears, depending on the function type.

F10 toggles between the detail format and tabbing sequence panels.

This panel allows you to see the sequence in which fields are displayed and if you are using ENPTUI, to adjust the sequence as you prefer. For further information, see the section on ENPTUI later in this chapter.



Note: You can modify I/O usage on this panel, which allows you to hide the field.

2. Change the format details. Select one of the following options to change the format details and press Enter. An explanation of each option is provided below.

   - I = input
   - O = output
   - H = hide
   - – = drop

   The panel is refreshed and your selection is reflected in the Ovr (Override) field.

Hiding/Dropping Fields

To hide or drop fields from your device design you need to change the device field entry properties of the device design fields.

Note: Only fields on the control key format or fields on the Prompt Record (PMTRCD) function type can be dropped. Record/detail fields can only be hidden. If the option is available for a field, it is generally better to drop a field rather than to hide one. Hidden fields generate associated processing while dropped fields do not. You can use access paths with dropped fields to achieve the same result.

Setting the Subfile End Indicator

The Subfile End (YSFLEND) model value controls whether the '+' sign or More. . .' appears in the lower right location of the subfile to indicate that the subfile contains more records. This capability is available for all subfile functions. The setting can be overridden with the associated function option. The possible values are:

- ***PLUS**—A '+' sign indicates that the subfile contains more records. This is the shipped default.
- ***TEXT**—'More. . .' indicates that the subfile contains more records. 'Bottom' displays to indicate that the last subfile record is displayed. Use of *TEXT prevents the last character of the last line of the subfile from being overridden by the '+'.

Existing functions default to *MDLVAL. To change to *TEXT everywhere, change the model value and regenerate your subfile functions.

Editing Device Design Function Keys

While you are in your device design, you can move or rearrange the order of the fields on your display using the following function keys.

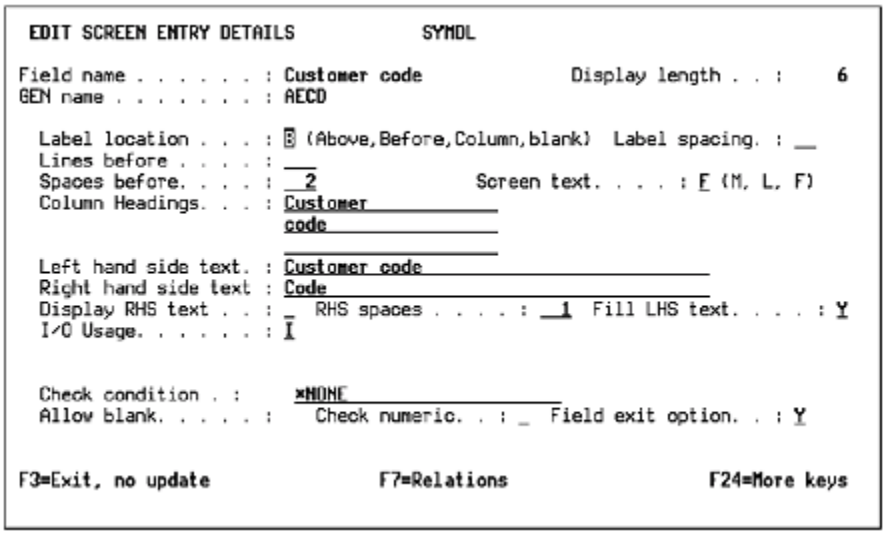
- F1 moves the field 40 positions to the left.
- F2 animates the panel using Toolkit.
- F3 exits the panel.
- F4 moves the field 40 columns to the right.
- F5 edits device format details of the format where the cursor is positioned.
- F6 cancels the pending operations.
- F7 displays the Edit Device Design Format Relations panel.
- F8 moves the selected field to the cursor position.
- F9 wraps text onto the next line starting from the field on which the cursor is positioned.
- F10 moves text one column to the right.
- F11 removes the line on which the cursor is positioned.
- F12 aligns text below the cursor position.
- F13 fast exits the panel.
- F15 moves panel window to the left margin.
- F16 moves window to the right margin.
- F17 displays a list of device formats.
- F18 displays the Edit Field Attributes panel.
- F19 adds new function fields to the device design.
- F20 edits the function field on the device design.
- F21 adds a line above the cursor position.
- F22 moves text one column to the left.
- F23 adds a constant field to the device design.
- F24 aligns all fields under the cursor position.

Modifying Field Label Text

You can modify the field details of the device using the following steps:

1. View the field text details. At the device design, place the cursor on the field you want to modify and press Enter.

   The Edit Screen Entry Details panel appears.



2. Modify the labels for the selected field.

Changing Display Length of Output-Only Entries

From the Edit Screen Entry Details you can override the *display* length of any output-only entry by entering a value in the Override length field. An entry is considered output-only if its I/O Usage is O.

Note: You can also use this method to override the model-wide length of the input capable *SFLSEL (subfile selector) entry for a function.

Override length field guidelines:

- The value you enter must be shorter than or equal to the actual length of the entry. The current display length is shown in the Display length field.
- If you enter 0 or leave the Override length field blank, the display length defaults to the value shown in the Display length field. For *SFLSEL, if the Override length field is 0 or blank, the display length defaults to the model-wide length.
- If data for the entry is longer than the display length, the data is truncated.
- If you change the I/O usage for the entry to I (input), you cannot change the entry's display length.

Displaying Device Design Formats

You can see which formats are present in a device design using the following instructions:

View the design formats. Press F17.

The Display Screen Formats panel appears.

Editing Device Design Formats

You can see which field entries are present in a device design format using the following steps:

1. Edit the formats. Place the cursor on the selected field on the format and press F5.

   The Edit Screen Format Details panel appears.

   Note: If there is more than one format, CA 2E shows you all choices. Select the format you want to edit.

2. Modify the format details.

Viewing and Editing Format Relations

Each of the database fields present on a function's device format is there because of a relation. Each relation in the access path to which the function attaches gives rise to a device design relation. Each of these relations is resolved into one or more field entries in the device format.

By default, all of the relations on an access path are present on the device design. Depending on the function type, you can override the defaults to drop particular relations. Dropping relations has the effect of dropping the panel field entries resulting from the resolution of the relation.

File-to-file relations, such as Refers to, can also lead to referential integrity checking. This check is implemented as a Read to the Referred to file in order to ensure that a valid key was specified. You can improve performance by dropping or using these checks for functions where this check is not required. This would typically be on those functions that do not update the associated foreign key. The Edit Screen Format Relations panel is used to adjust how these relations are processed.

To edit the format entries, use the following steps:

1. View the details. At the device design, place the cursor on the field you want to modify and press Enter.

   The Edit Screen Entry Details panel appears.

2. View the format relations. Press F7 to view the relations.

   The Edit Screen Format Relations panel appears.

   Note: Use the Edit Report Format Relations panel for report designs, which you reach through the Edit Report panel.

When CA 2E generates a program to implement the function, it normally includes source code to check that all of the relations are satisfied. For a particular relation on the device design you can specify that this enforcement of the device design relations should not take place. You can specify five different degrees of enforcement:

1. Required Relations

The relation is always enforced. Any field arising from the relation must be entered with a non-zero or non-blank value. If the relation is a file-to-file relation, a record must exist for each field on the referenced file.

2. Optional Relations

The relation is only enforced if a value is entered for any of the fields that resolve the relation.

3. Dropped Relations

The relation will be dropped. The field arising from the relation is omitted from the format altogether. Dropped relations are only allowed for Print File and Prompt Record functions.

4. User Relations

The relation will not be checked. Fields arising from the relation are still present on the format. You can add your own validation for the relation at an appropriate point in the action diagram.

5. No-Error Relations

The relation is checked (under the same conditions as for optional). If no record is found, no error is flagged. The relation is used only to retrieve information, if present.

The following table identifies the relations and how they can be set using the Edit Screen Format Relations panel.

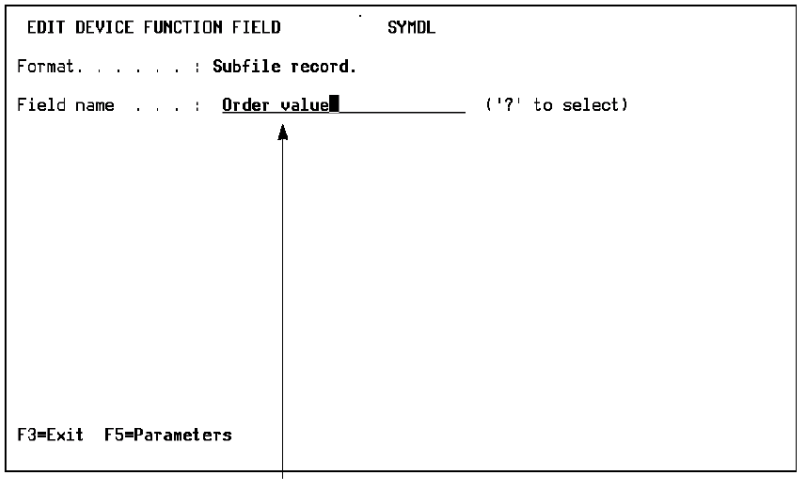
Function Type	REQUIRED	OPTIONAL	DROPPED	USER	NO-ERROR
PMTRCD	Any Rel	Any Rel	Any Rel	Any Rel	File to File
DSPRCDn	Any Rel		None	Any Rel	
EDTRCDn	Any Rel	Non-key	None	Any Rel	File to File
SELRCD	Any Rel		None	Any Rel	
EDTFIL	Any Rel	Non-Key	None	Any Rel	File to File
SELRCD	Any Rel		None	Any Rel	
DSPFIL	Any Rel		None	Any Rel	
DSPTRN (2)	Any Rel	Non-key	None	Any Rel	File to File
EDTTRN	Any Rel		None	Any Rel	

Adding Function Fields

You can add or edit function fields to the device designs of functions using the following steps:

1. View the function fields. At the device design, place the cursor to the left of where you want to add the function field and press F19**.**

   The Edit Function Field panel appears.



2. Add the function fields. Type the name of the field that will be added or choose the selected function field from a list by entering: **?**.

If you are adding a new function field, use ? to display all fields and F10 to create a new one. During this selection process, you can define any new function field for use in this or any other function.

Modifying Function Fields

You can change the definition of an existing function field while in the device design. Remember that this changes the definitions of the field for all functions using this definition.

1. At the device design, place the cursor on the selected field and press F20.

   The Edit Function Field panel appears.

2. Type a **?** next to the field name and press Enter.

   The Display Fields panel appears.

   Note: If you know the name of your function field you can type it after the ?. This positions the cursor on that function field when the panel appears.

3. Zoom into the field and modify the edit field details.
4. Press F3 to exit the field details and return to the device design.

Deleting Function Fields

You can remove a function field from the device design using the following steps:

1. At the device design, place the cursor on the selected function field and press F20.

   The Edit Function Field panel appears.

2. Delete the function field. Press F11.

   Note: Deleting a function field immediately removes the field from your device design, even if you exit the device design without saving your changes.

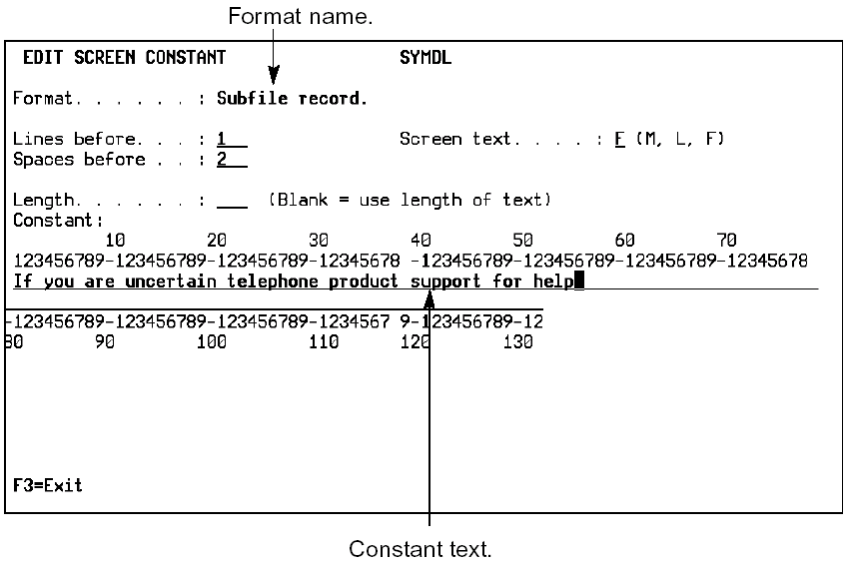
Deleting a derived function field and exiting without saving does not remove the function field from the action diagram.

Adding Constants

You can add constants to the device format using the following instructions:

Add the constant. At the device design, place the cursor on the field after which the constant is to appear and press F23.

The Edit Screen Constant or Edit Report Constant panel appears.



Deleting Constants

You can remove a constant using the following instructions:

Delete the constant. At the device design, place the cursor on the constant press Enter, and then press F11. The Edit Screen Constant panel appears.

Note: Deleting a constant immediately removes the constant from your panel design, even if you exit the panel design without saving your changes.

Modifying Action Bars

Depending on the function, CA 2E provides logical action bar defaults for action bar choices, action bar mnemonics, pull-downs, and associated descriptive text, and pull-down accelerator keys.

The standard (*STD CUA) action bar header/footer default choices are described next.

CUA Text Standard Action Bars

Depending on the function, CA 2E provides logical action bar defaults for action bar choices, action bar choice mnemonics, pull-downs and associated descriptive text, and pull-down accelerator keys.

The *STD CUA Action Bar header/footer default choices are described below.

File

The File choice is for actions that apply to the primary conceptual object to which the panel applies. The following actions are used for one or more panels:

Action	Meaning
New	Switch to add mode
Open	Switch to change mode
Reset	Reset panel
Delete	Delete the object instance
Cancel	Return to previous panel
Exit	Leave without update

Function

The Function choice is for actions that apply to the whole panel or interface object. When the action bar definition panel is loaded, each *CMDKEY condition that is referenced in the function's action diagram and not already in the action bar definition is loaded as a Function pull-down choice. The following action is used for one or more panels:

Action	Meaning
Actions	*CMDKEY conditions loaded from the action diagram

Selector

The Selector choice is for actions that apply to a part of the panel or interface object. When the action bar definition is loaded, each *SFLSEL condition referenced in the function's action diagram and not already in the action bar is loaded as a Selection choice. The following actions are used in one or more panels:

Action	Meaning
Delete item	Delete the defined item or items
Selectors	*SFLSEL conditions loaded from the action diagram

Help

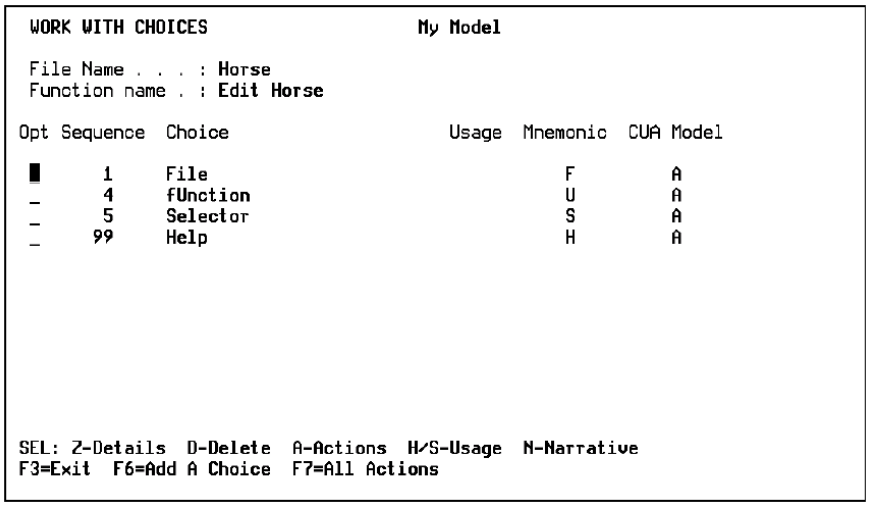
The Help choice is for actions associated with Help. The following actions are used in one or more panels:

Action	Meaning
Help	Help for the part of the panel where the cursor is located
Ex help	Help for the whole panel
Keys help	Help for accelerator keys
Help index	Help index

You can override the action bar defaults, using the Action Bar Editor and the following instructions.

1. Go to the Action Bar Editor. Move the cursor to the action bar at the top of the panel and press Enter.

   The Action Bar Editor appears.



2. Select the option you need to make your modifications. The first panel, Work with Choices, allows you to modify action bar choices. From this panel, you can access other panels on which you can modify the pull-down choices. These panels are as follows:

   - **A**—Work with Actions of a Choice
   - **F7**—Work with Actions
   - **Z**—Edit a Choice (panel displays details)
   - **N**—Edit Narrative

Modifying Windows

A window can range in size from 5 columns by 5 rows, not including the borders, to just less than the full size of the display. When implementing DDS windows, i OS adds two columns on each side of your window and one additional row. The columns protect the display attributes of the underlying panel. The additional row is reserved for i OS system messages. Therefore, the application program window appears slightly larger than when viewed in the panel design.

Attribute	Values
Width	76

Attribute	Values
Depth	22
Location	A for Auto (the program automatically places the window). The other option for Location is U (the window location is defined by the programmer). To locate the window, use the Row, Column, and Corner settings. The corner that you specify is placed in the displacement specified from the top left of the screen.
Row	1
Column	1
Corner	TL

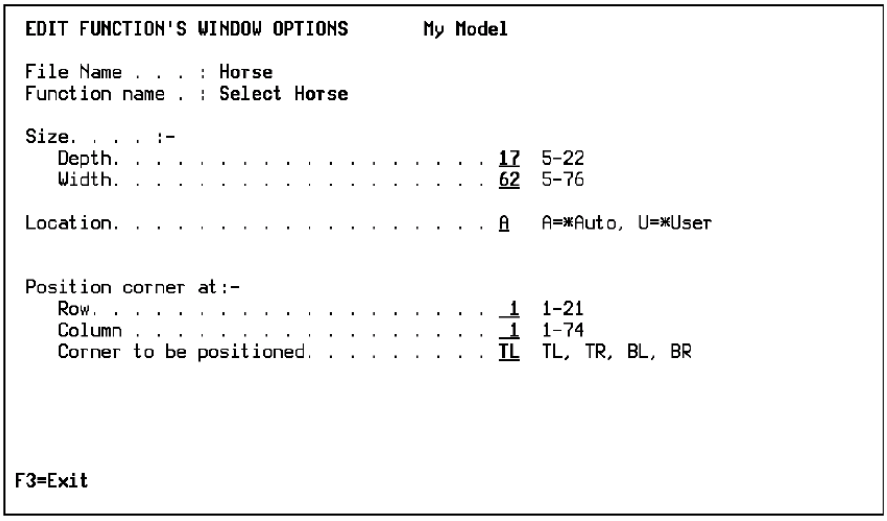
In the following example, the window's displacement is specified by naming the row and column at which a particular corner is to be positioned, for example, top left (TL), top right (TR), bottom left (BL), or bottom right (BR).

Note: You do not see the displacement when editing the device design.

To change the window size and other window features, use the following steps:

1. Go to the Windows Options. From the device design, place the cursor on the function title and press Enter**.**

   The Edit Function's Windows Options panel appears.



2. Modify the defaults to meet your requirements.

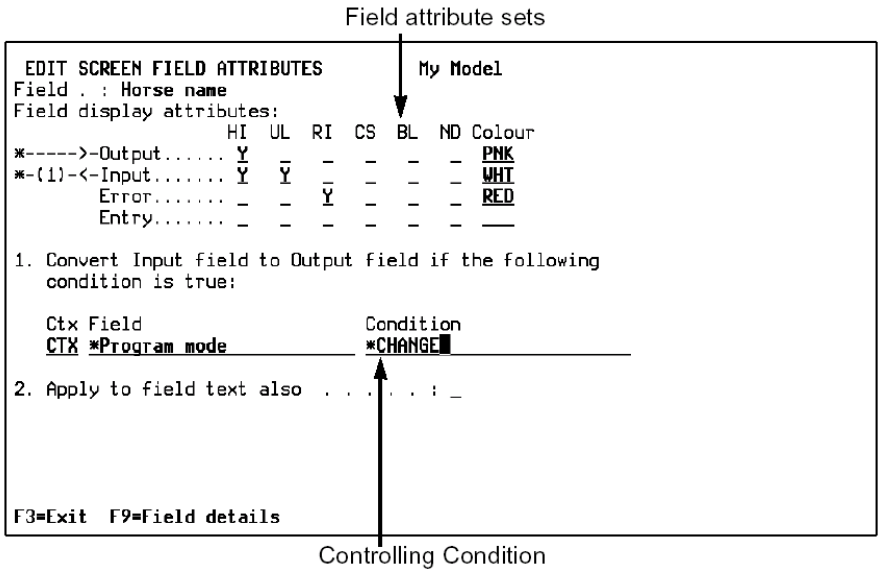
Modify the defaults to meet your requirements. Modifying Display Attributes and Condition Fields

The display attributes of a field are initially set by default to the model standards. These attributes define how a field is displayed; for example, if it is input, output, or in error. You can change the model standards and the controlling condition using the Edit Default Device Field Attributes (YEDTDFTATR) command.

The display attributes for an individual field on a panel can be changed using the following steps:

1. At the device design, place the cursor on the selected field and press F18.

   The Edit Screen Field Attributes panel or Edit Report Field Attributes panel appears.



2. Modify the attributes or conditions.

Note: If you enter Y for Apply to field text and if the condition to display the field as output is true, the LHS text, RHS text, or column heading associated with the field assumes the output field attributes you assigned for the field.

Fields can be switched to another set of display attributes depending on the condition. This includes switching to output only or to non-display. If you need to condition the field based on a complex or compound condition, you should consider adding a derived function field that sets a true/false condition field to the panel as a non-display field.

This change should yield a true/false status condition. The associated action diagram can contain a compound condition. Using this technique, any combination of parameters and conditions can be encapsulated as a single status condition and used to condition a panel field.

Usages	Display	Printer
O	Y	Y
I	Y	-
B	Y	-
H	Y	Y

Display Attributes	Display	Printer
HI	Y	–
UL	Y	Y
RI	Y	–
CS	Y	–
BL	Y	–
ND	Y	Y

Editing Panel Design Prompt Text

Panel designs can include two types of explanatory text:

- Explanations of the standard function key meanings (such as, F4 prompt, under the CUA Entry standard)
- Explanations of the standard subfile selector value meanings (such as, 4-Delete on CUA Entry, Delete is an action bar choice on CUA Text)

An initial version of this text is automatically built for each device design from the action diagram of the device function. You can modify it if needed.

The way the explanation text strings are provided and the order in which they are placed on panel designs depends on the interface design standards you use.

Function Key Text

Function key selection text is always placed in the position specified for the *CMDTXT fields on the standard header function Define Screen Format (DFNSCRFMT) associated with the function of the device design you are editing. You can change the current standard header for the function using the Function Options panel. One or two lines of text can be specified. If two lines of text are specified, but only one is needed, the text is placed in the lower of the two lines.

You can control where the text appears by changing the usage of the *CMDTXT1 and *CMDTXT2 fields on the associated standard header function’s device design.

Subfile Selector Text

Subfile selector value explanatory text is built only if there are subfile selector values for the function.

You have a choice of two different positions in which to place any subfile selection explanatory text:

- Place the text as a separate line on the subfile control record (CUA standard).
- Place the text as part of the function key explanation text. Typically at the bottom of the display (System 38 standard).

To manually change the text, use the following step to refresh the text from any action you take in the action diagram:

Place the cursor on any line of explanatory text and press Enter. The Edit Command Text panel appears.

The position at which subfile selector value explanatory text is placed depends on whether you specify that the *SELTXT fields on the subfile control fields display on the associated standard header/footer.

1. If the *SELTXT fields (up to two lines are permitted) are visible. The descriptive text appears at the position indicated on the subfile control format.
2. If the *SELTXT fields are hidden, the selection value explanation text appears at the position indicated by the *CMDTXT fields as specified by the associated standard header function. If there is only one line of *CMDTXT, the subfile explanation text appears on the same line. If there are two lines of *CMDTXT, subfile explanation text appears on the first line and the function key explanations on the second.

   Note: If you hide these fields on any device design then they are hidden and the associated text is not moved or adjusted.

Whether or not the *SELTXT fields are available on screen designs is controlled by the Enable Selection Prompt Text function option of the associated standard header function and the usage (hidden - H or output - O) of the *SELTXT fields on the device display.

The selection text fields (*SELTXT1 and *SELTXT2) can be preceded by a third field, the selection prompt field (*SELPMT), which tells you how to use the explanation fields.

For example: Type **option**, press Enter.

Selector Role

For fields that are selectors on the control formats of DSPFIL and SELRCD functions, you can specify the nature of the selection in terms of a relation operator. This can be one of the following:

- **Relational Operator (EG, NE, LT, LE, GE, GT)**—Selects records with field values that satisfy a test specified by the operator and the selector field value.

  For example: GT 10, EQ IBM, LT 100

- **Start Operator (ST)**—Selects records with field values that start with the specified value (character fields only).

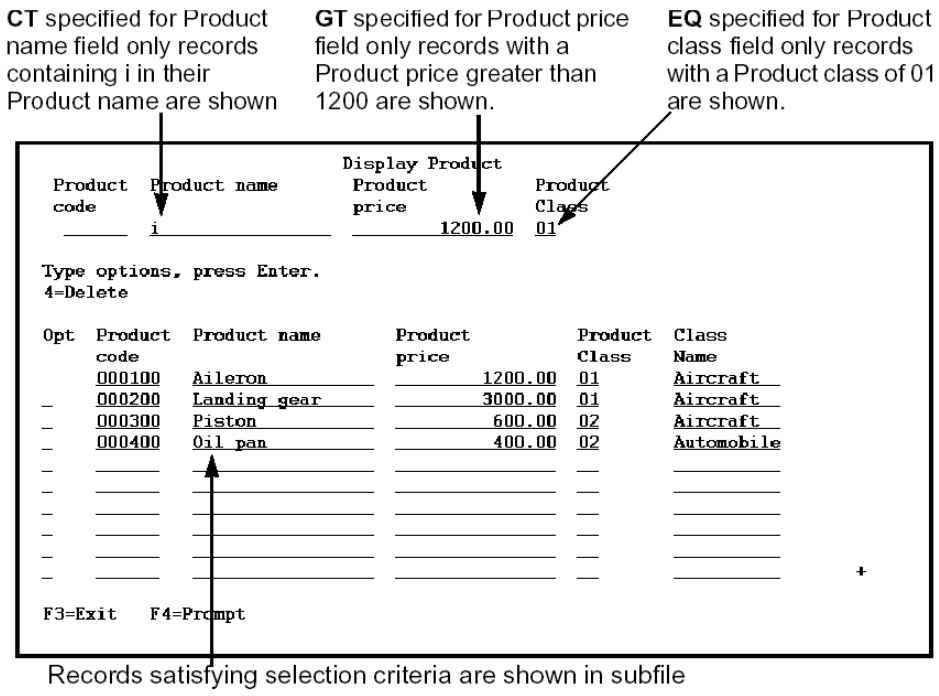
  For example: A selector field value of TXT would select TXTDTA and TXTSRC but not QTXTSRC.

- **Contains Operator (CT)**—Selects records with field values that contain the specified value for the field (character fields only).

  For example: A selector field value of TXT would select QTXTSRC, FREDTXT, and TXTDTA.

If a value is entered for the selector field, selection is only applied at execution time. If a value is entered in more than one selector field, the selection criteria are logically ANDed together.

For example, on the following DSPFIL function example, three different types of selector role are specified.



Add SFLFOLD/SFLDROP to a Subfile Function

You can automatically add Subfile Fold (SFLFOLD) and Subfile Drop (SFLDROP) functionality to your generated subfile functions. SFLFOLD/SFLDROP lets you define a command key for the function so that it can be used to toggle between folded mode and dropped mode. In folded mode, the subfile displays as it does in the screen designer, with each subfile record taking up more than one display line. In dropped mode, the operating system automatically truncates each subfile record so that only the data fields that appear on the first display line are displayed. This means that twice as many subfile records are displayed.

SFLFOLD/SFLDROP functionality is only applicable to multiline subfiles. That is, subfiles where one subfile record extends over more than one line on the screen. It is available for the following function types:

- DSPFIL – Display file
- DSPTRN – Display transactions
- EDTFIL – Edit file
- EDTTRN – Edit transactions
- SELRCD – Select record

Follow these steps:

1. Edit the device design for a subfile function.
2. Verify that the subfile record extends over more than one display line.

3. Press F17 to display the DISPLAY SCREEN FORMATS panel and select option Z (Details) against the Subfile Control format to display the EDIT SCREEN FORMAT DETAILS panel.

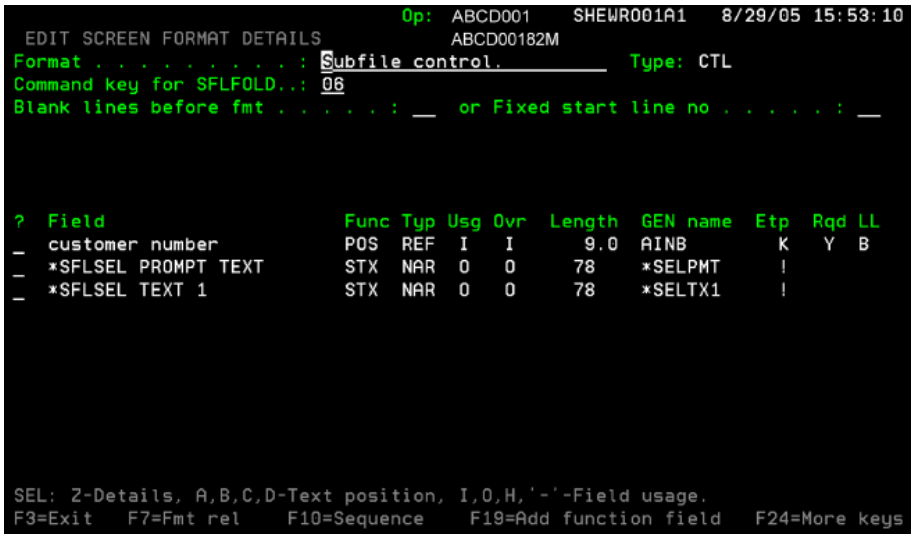
4. Enter the desired command key to use for SFLFOLD/SFLDROP, toggling in the Command Key for SFLFOLD field.

   Valid values are 03, 05–11, or 13–24. All other keys are reserved for other use by CA 2E.

   Notes:

   - It is your responsibility to ensure that the selected command key does not interfere with a command key that is already in use for this screen. The 05 command key is usually selected for the *CMD key field *Reset list condition. However if 05 is not used in the *Reset list condition, then the 05 command key is available to be used for SFLFOLD/SFLDROP toggling.
   - If this field is left blank, SFLFOLD/SFLDROP functionality is not enabled for this function. Because automatic checking is not done to ensure that the selected command key is not used by the function, it is your responsibility to ensure that the specified SFLFOLD/SFLDROP key does not clash with a command key used by the function.

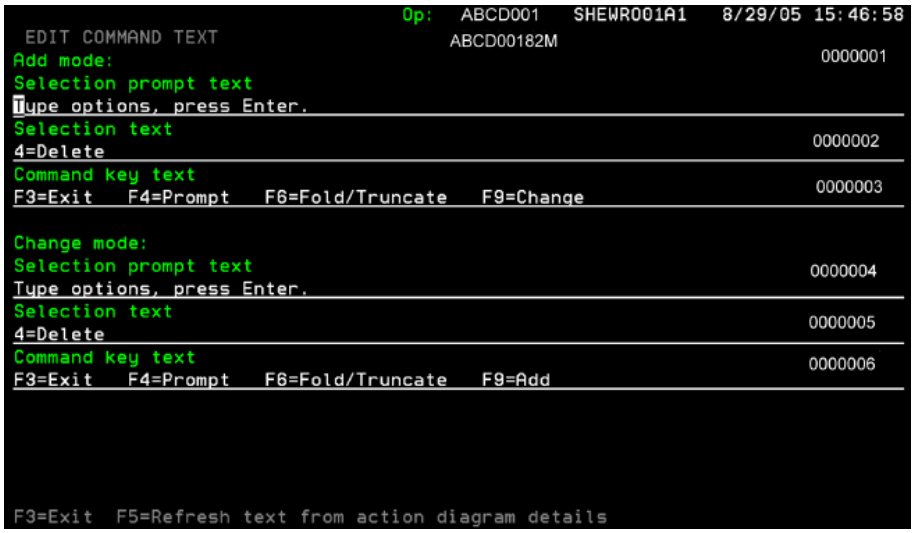
   In the following example, 06 has been chosen as the SFLFOLD/SFLDROP command key:



5. Return to the main screen design page, move the cursor to the command key text, and press Enter to display the EDIT COMMAND TEXT panel.

6. Press F5 to refresh the display. The text for the specified command key is added automatically.

   Edit or remove the text: the Fold/Truncate text is retrieved from message identifier Y2F5361 in message file Y2ALCMSG, which can be edited to globally change the default text. Whether this text is displayed or not, the SFLFOLD/SFLDROP functionality is still enabled for the function.



   When the function is generated, code is automatically included in the DDS for the display file and in the source for the program to set and check the subfile mode before and after the screen is displayed. The subfile mode is available programmatically through the *Subfile mode field in the PGM context. It has an internal DDS name SFM, and has the following conditions:

```
*Folded              VAL     0       0
*Truncated  VAL      1       1
```

7. Within the action diagram, setting the PGM.*Subfile mode field prior to the screen being displayed causes the screen to be displayed in the desired format (folded or dropped). By default, a screen displays in folded mode. Pressing the specified SFLFOLD/SFLDROP command key when the screen is displayed does not return control to the program but simply causes OS/400 to redisplay the screen in the alternate format (folded or dropped). Upon return from the display, because you have pressed another command key or the Enter key, the value of the PGM.*Subfile mode field can be rechecked to see if the mode has been changed since the screen was displayed.

Example SFLFOLD/SFLDROP

In the previous procedure, when the screen is initially displayed, it displays as follows, where only four records are displayed, each showing the full subfile record:

Pressing F6 redisplays the screen, showing only the first line of each subfile record, with up to 12 records partially displayed. The following panel demonstrates this screen:

ENPTUI for NPT Implementations

The enhanced non-programmable terminal user interface (ENPTUI) within CA 2E, provides options for:

- Menu bars
- Drop-down selection fields
- Cursor progression
- Entry field attributes
- Multi-line entry
- Edit mask

The environment required to implement ENPTUI features includes:

- V2R3 of i OS or V2R2 of i OS with PTFs.
- An InfoWindowII workstation with a workstation controller that supports an enhanced data stream. Alternatively, V2R3 of Rumba/400 supports the ENPTUI features. ENPTUI functions are ignored or display with varying degrees of quality on other terminals.

For more information on display and terminals, see IBM's Manual *Creating a Graphical Look with DDS* SC4101040.

Creating Menu Bars

You can generate DDS menu bars instead of CA 2E Action Bars with ENPTUI. To indicate whether to generate the run-time action bars or DDS menu bars for NPT, use the model value:

YABRNPT

This model value is used for NPT generation only and can be overridden at the function level with function options. The Panel Design Values panel from the Display Model Values panel (YDSPMDLVAL) displays this model value.

Valid values are:

*ACTBAR = Create CA 2E Action Bars, default for models created prior to COOL:2E Release 5.0.

*DDSMNU = Create DDS Menu Bars, the default for models created after COOL:2E Release 5.0.

You are advised to migrate to DDS Menu Bars if you have upgraded a model from a release of COOL:2E earlier than 5.0. This is because DDS Menu Bars make use of the new i OS ENPTUI features that allow the menu bars to be coded in the DDS for the display file. The CA 2E Action Bars require an external program to be called to process the action bar. As a result, the DDS Menu Bars are faster, have more functionality, and create more efficient CA 2E functions.

Note: If you use DDS menu bars, you need to use the User Interface Manager (UIM) to define help for your actions and choices.

The Edit Function Option panel has the following function option:

If action bar, what type?: (M-MDLVAL, A-Action bar, D-DDS menu bar)

The default value is M, representing MDLVAL.

For DDS Menu Bar:

- Choices and actions are white
- The color of a selected choice or action is the same color as the menu bar separator line
- The color of the menu bar separator line is the color assigned in YWBDCLR (Window border color)
- A separator line is required for menu bar generation
- If a choice has no actions attached, it does not display

Unlike CA 2E Action Bars, menu bar selections may not have the same sequence number. CA 2E currently allows NEW and OPEN to each have a sequence number of 1. It is not required that this be changed. However, if you do not change it, the generators reassign the sequence number to 2 for NEW.

Assigning Sequence Numbers for Actions

An example of sequence number assignments from the Work With Actions of a Choice panel follows:

```
Opt     Sequence     Action     Usage     CUA Model

            1        Open                   A
            2        New                    A
            3        Reset                  A
            99       Exit                   A
```

Gaps in the numbering sequence of actions create a blank line between action choices in the pull-down menu. For example, the menu resulting from this example displays a gap between the Reset and Exit choices. This i OS feature allows you to group options, but you can eliminate an unwanted gap by renumbering the actions sequentially.

Working with Choices

If there are more choices than can display on a single line, they wrap to the next line. The display menu choices can take up as many lines as needed. The remainder of the device design is displaced downwards. However, CA 2E Action Bars horizontally scroll the choices without affecting the remainder of the device design.

- The maximum number of lines that can be devoted to the display of choices is twelve including one line for the menu bar separator
- The minimum number of lines required for a DDS Menu Bar is three
- CA 2E Action Bars require two lines for display

Note: If you change a function from action bar to menu bar it may fail to generate due to the displacement of the device by the additional lines required for the menu bar display.

Specifying a Drop-Down Selection Field

When a STS field is assigned as a drop-down list, a Window is created in the display file. The window contains a single choice field (SNGCHCFLD radio button selection), where the choices for the control equal the available condition values.

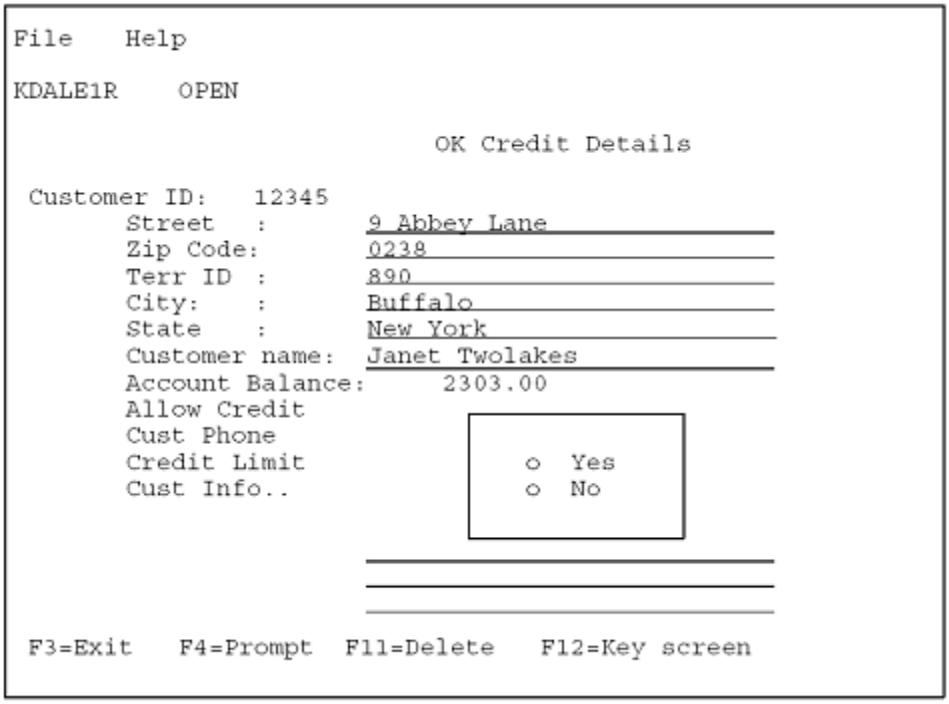
Note: If the number of values attached to a condition changes, you must regenerate the function.

The condition name associated with each condition value appears in the pop-up window. After you select a value, the window closes and the external value for the selected condition is placed in the prompted field.

The drop-down list is positioned on the screen with its top left corner in the first character of the status field. If there is not enough room on the screen, the drop down list moves left and up until it fits on the screen.

Note: The drop-down list can only be prompted using F4 (or equivalent assignment); you cannot prompt with a question mark.

The following is an example of a drop-down list:



When you prompt Allow Credit with the F4 key, the previous window displays and overlays other fields. When a value is selected, the window is removed.

You can designate a STS (status) field as a drop-down list at the condition level, the field level, and the screen entry level, as in the partial screen examples that follow:

```
EDIT LIST CONDITION                              MYMDL
Field name.....: Allow Credit          Attr. :   STS
Condition name:.....: *All values                Condition No. : 1100258
Condition type:.....: LST
Prompt.....: Drop Down List
```

```
EDIT FIELD DETAILS                               MYMDL
.
.
.
Control.....:
Default condition.....:*None
Prompt.....: Drop Down List
```

```
EDIT SCREEN ENTRY DETAILS                        MYMDL
.
.
.
Prompt.....:  Drop Down List
Check condition.....      ...:*ALL values
Allow blank.....: Y                                Field exit option:
```

- Condition level is the highest level from which you can specify a drop-down list with inheritance by every field that shares this condition
- Field level assignments override condition level assignments
- Screen level assignments override field level assignments

On the Field level, F10 toggles between the field control information and the appearance fields.

To specify,

1. From the Edit List Condition panel, the Edit Field Details panel or the Edit Screen Entry Details panel, place your cursor in the Prompt Type field.
2. Enter ***DDL** to specify a Drop-Down List.

Note: The Prompt Type field only displays if the field is STS.

The field may be prompted with a ? to get a list of valid values. Valid values for Prompt Type at the Condition, Field or Screen level are:

*Drop Down List (or *DDL)

*Condition Values Displayer (or *CVD) is the

default value

Defaulting of Prompt Type

If you do not specify a Prompt Type from the device design:

- The default value is the value assigned at the field level
- If there is no value assigned at the field level, the default is taken from the condition level assignment
- If no condition level assignment is set, the default prompt is the Condition Value Displayer

The Prompt Type displays as:

- Normal intensity when displaying the default assigned at a previous level
- High intensity if the default is overridden

Some Specifics of Drop-Down Lists

Some things to keep in mind when specifying a drop down selection list are:

- Status fields with more than 22 condition values do not generate as drop-down lists, since they are too large for the display.
- There is a maximum of 99 drop-down lists associated with a single program.
- For EDTRCDn and DSPRCDn functions, if the same status field appears on more than one screen, then all instances should be the same prompt type. That is, either all condition value displayer or all drop-down list. Each instance of the field should also be assigned the same check condition whether condition value list or drop-down list is used.

Mnemonics

You can assign mnemonics using the Edit Field Condition Details panel. They are also displayed on the Edit Field Conditions panel. When assigning a mnemonic, note that:

- The character chosen as the mnemonic must be a single-byte character present in the condition name
- The character must not be duplicated within the same status field
- Blank and greater than (>) characters cannot be used as mnemonics
- DBCS mnemonics are not supported by DDS

National Language

For national language applications (NL3), the condition value text is translatable at run time. The size of the Drop-Down List window is large enough to hold the maximum length condition value name, which is 25 (for nonNL3 applications, the window is only as wide as is necessary to display the longest condition value name within the given check condition). Just as with the condition value displayer, the condition values file must be translated. Mnemonics are not available in national language applications unless translators embed the mnemonic indicator '>' in the translated text. If this is done, the translator must be careful not to duplicate mnemonics within a single check condition.

Assigning Cursor Progression

Cursor progression assignments give you the ability to assign a tabbing sequence between fields that are logically grouped together within generated functions. For each field within a record format, you can assign its position within the tabbing sequence. For back tabbing, cursor progression assignments are reversed.

Note: The last field displayed within a format should be the highest field in the tab sequence order. This is a design suggestion, because if not designed in this way, tabbing within the generated function loops within a single format.

To edit the cursor progression assignments, press F10 from Edit Screen Format Details panel. This panel is shown earlier in this chapter in the Editing Device Designs section.

You can assign tabbing sequence and change the order in which fields display from the Edit Screen Format Details panel. "F8 = move", from the device design editor, was the only way to change the order in which fields displayed. When you use F8 to move a field, that field is assigned a new display sequence number equal to that of the target field plus 0.01. Therefore, if you key in new display sequence numbers, you should key in only integer values.

Cursor Progression and Subfiles

For RCD context, valid values for tabbing sequence number are 0 and 1.

- Value 0 means that no specific tabbing requirements are to be generated and the cursor should progress to the next field in the same subfile record.
- Value 1 means that when TAB is pressed in the indicated field then the cursor should advance to the same field in the next subfile record.

All functions are created with default tabbing sequence numbers of 0 to indicate default tabbing: left to right, top to bottom.

Setting an Entry Field Attribute

The Entry Field Attribute allows attributes to be assigned to a field and used when the cursor enters that field by tabbing or with a pointing device (mouse). You can only set these attributes for input capable fields. They are maintained on the Edit Screen Field Attributes panel shown earlier in this chapter, in the row labeled Entry. The default setting is blank for no attribute change.

Note: Entry Field Attributes are ignored if the field is converted to output or, at run time, if the first character in the input field is protected by an edit mask.

You can use the YEDTDFTATR command to set a default for Entry Field Attribute.

Assigning Multi-Line Entry

Any alphanumeric field with a length greater than one may be designated as a multi-line entry field (MLE). MLEs are long text fields that are wrapped into a text box. Assignments made at the field level are inherited by all screen entries based on this field. Multi-line entry fields cannot be specified for STS type fields or for any field in the RCD context, that is, it cannot be used on a subfile.

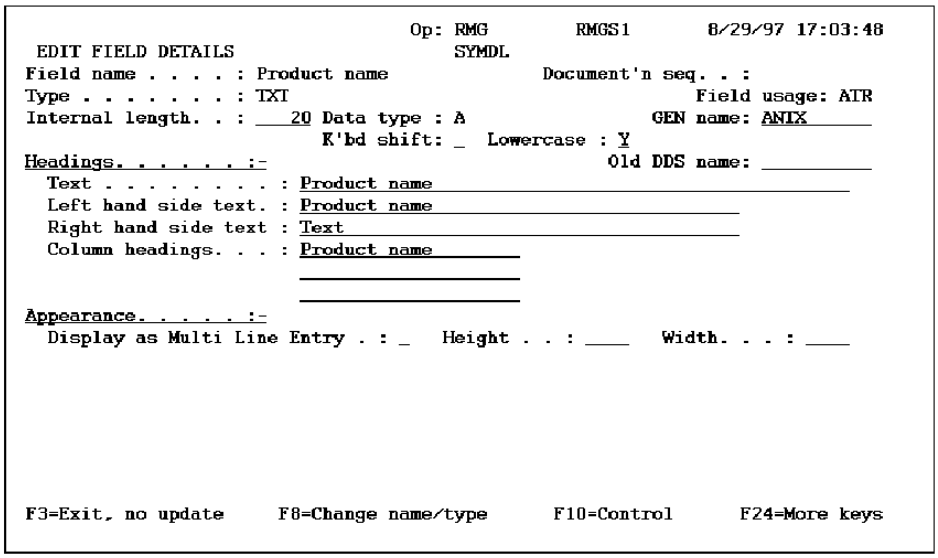
Examples of how to specify that a text field display as an MLE follow.

1. From the Edit Field Details - Field Control panel, press F10 to access the appearance panel. This panel shows the line:

   Display as Multi Line Entry: Height: Width:

   The multi-line entry field displays in normal intensity if it uses the default value from the field definition and displays in high intensity if the default has been overridden.

2. Specify either height or width. When you specify one, the other is calculated for you. Height is the number of rows; width is the number of columns.



If a field is an MLE, then on the device design editor the field displays with a length equal to the column width assigned to the MLE.

All parts of the MLE display on the device design editor; however, it is the responsibility of the designer to ensure that no fields are overwritten by the additional lines of the MLE. Positioning of other fields is relative to the first line of the multi-line entry field.

For workstations that are not attached to a controller that supports an enhanced data stream, the different parts of the continuation field are treated as separate fields for editing, but when enter is pressed, a single field is returned to the program.

Using an Edit Mask

Edit Mask allows input capable fields to contain format characters that are ignored. For example, in the formatted phone number field, "(_ _ _)_ _ _-_ _ _ _", when the user enters data, the cursor skips the protected format characters. If Mask input is set to Y then each non-blank character in the selected edit code is masked. The value 0 is not masked. If the field containing the edit mask is initially displayed with no data, then the masked characters do not display but the cursor still skips over the protected positions. When the field is redisplayed with data, the masked characters appear.

If the first character in an EDTWRD is a format character, as in the phone number example, that character does not display for NPT DDS. This is a limitation of DDS that can be corrected by allowing for an integer to the left of the left-most format character, for example, '0( ) - '; in this case you also need to increase the field length by one. Alternatively, you can create your own EDTCDE definition using the i OS Create Edit Description (CRTEDTD) command.

Note: Do not mask fields that represent a quantity or value. If an attempt is made to mask a numeric quantity, unpredictable results occur. For example, masked characters are not returned to the application, so the decimal place is lost if it is masked. The edit mask feature is intended for numeric fields that are always formatted in the same manner, such as phone number, social security number, part number, date, and time.

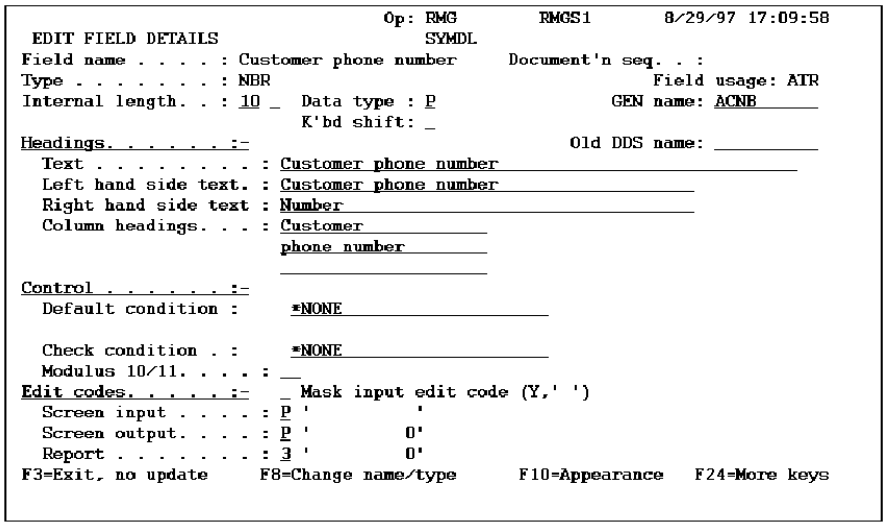
The edit mask option is maintainable at the field and screen levels.

Note: The edit mask option is ignored when the workstation is not attached to a controller that supports an enhanced data stream.

1. Set the field level value for the mask input option on the Edit Field Details panel.
2. Enter **Y** in the field:

   Edit Codes.....: Mask input edit code (Y,' ')

   The default value is blank, for do not mask. F10 toggles between the field control information and the appearance fields.



3. The Edit Screen Field Attributes panel is used to override the field level assignment.
4. To reset to the field level default, enter a blank for Mask Input and press Enter.

The data displays in:

- Normal intensity if it is displaying the default assigned at the field level
- High intensity if the default is being overridden

Edit Mask - ZIP + 4 Example

To edit Zip+4 zip codes with a hyphen in front of the additional four digits:

Specify an EDTWRD in YEDTCDERFP with the format:

'_ _ _ _0-_ _ _ _'. This displays as: '_ _ _ _ _-0000'.

If you indicate that this field should be masked, all blank characters, except 0, are masked. The cursor jumps over the - as data is keyed into the field. Without the 0 in the EDTWRD definition, the zip 12345- would display as 1-2345.

Editing Report Designs

CA 2E provides a default device design for printed reports. However, you can modify the device designs to suit your specific needs. The following topics provide you with information on how to modify the report structure and formats.

CA 2E provides you with two distinct steps in defining a report. They are:

- Specification of the individual report design. A default report design is provided automatically for each print function, such as each PRTOBJ or PRTFIL function that you define. You can then modify the design.
- Inclusion of the individual report designs within an overall device structure. You specify how this is to be done by explicitly connecting the functions together and specifying the parameters to pass between the functions.

You can modify the device design at the format and field level:

- Modify the device design at the format level to
  - Suppress (by either dropping or hiding) formats
  - Modify the spacing between formats
  - Specify whether the format is to be reprinted on overflow
  - Modify the format indentation
- Modify the device design at the field level to:
  - Rearrange the order of fields
  - Drop fields or field text
  - Add extra fields and or text
  - Modify field text

Note: You cannot override the output field length.

Standard Report Headers/Footers

The top and bottom lines of the report designs are defined by the DFNRPTFMT function. The name of the standard header function associated with each function is shown on the Edit Function Options panel. Unless you explicitly selected a particular header/footer, this defaults to the default header/footer function. The shipped standard report page header is 132 characters wide, but can be changed to an alternate width.

Understanding PRTFIL and PRTOBJ

There are two different types of report functions: Print File(PRTFIL) and Print Object(PRTOBJ).

- The PRTFIL function is an external function type that specifies a complete report in itself
- PRTOBJ is an internal function type that specifies a segment of a report for inclusion within another report function.

Note: Page headings and footers are always defined by the PRTFIL function.

PRTFIL

Each PRTFIL function specifies the layout and processing you use when printing records from an access path. It specifies the indentation of any embedded PRTOBJ functions within the overall report design. It specifies all global properties of the report such as page size and page headings.

- A print file function must be based on a Retrieval(RTV), Resequence(RSQ), or Query(QRY) access path.
- Calls to embedded PRTOBJ functions are inserted at the appropriate points in the action diagram of the embedding function.
- Each PRTFIL function has its own default report design that can be edited. Embedded PRTOBJ report designs are shown as part of the PRTFIL report design but are protected and cannot be edited directly. Only the indentation (distance of embedded design from the left-hand margin of the report) can be changed.

A PRTFIL function and its embedded PRTOBJ functions are implemented as a single i OS printer device file (PRTF) plus a single HLL program.

PRTOBJ

Each PRTOBJ function specifies the layout and processing you use when printing records from an access path.

- A PRTOBJ function must be based on a Retrieval (RTV), Resequence (RSQ), or Query (QRY) access path.
- The access path can contain several key fields, giving rise to several formats in the report design layout, and several key breaks in the logic of the action diagrams for the function.
- Each PRTOBJ function has its own default report design that can be edited.
- Page headings are not specified.

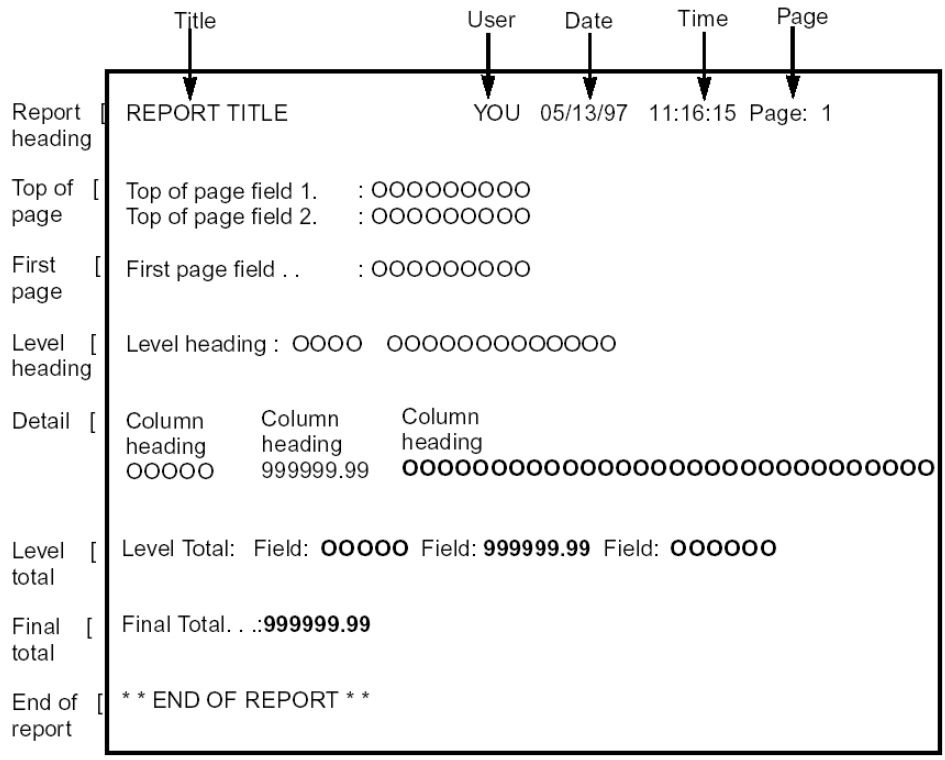
Modifying Report Design Formats

Report designs are made up of report design formats. When you first create a report by nominating an access path and a report function type, CA 2E automatically defines the appropriate formats for the report design. There are eight different report design formats. A typical report includes some or all of the following formats.

- The Report Heading Format—Includes the title, originator, and page information, defined by a Define Report Format (DFNRPTFMT) function. There is one Report Heading format per Print File function. This appears at the beginning of the report and on each page of the report. The information on this format is common to all report designs that share the same DFNRPTFMT function. This format cannot be dropped or hidden.
- The Top of Page Heading Forma—Includes the information to be repeated on each page. By default, this format is empty. There is one Top of Page Heading format per PRTFIL function. The information on this format is specific to the individual report. This format is not available for PRTOBJ.
- The First Page Heading Format—Includes the information to be printed before the first level heading or detail records for a given print function. By default, this format is empty. There is one First Page Heading format per report function (PRTOBJ or PRTFIL) only printed once per report function.
- The Level Heading Format—\-Contains fields appropriate to the key level. There is one key field per report function (PRTOBJ or PRTFIL). By default, this format is printed before the first detail or subheading within a level break. If the access path contains only one key field, this format is omitted.
- The Detail Format—Contains fields from the based-on access path. There is one Detail format per report function (PRTOBJ or PRTFIL). By default, this format is printed for every record read. This format cannot be dropped, only hidden.
- The Level Total Format—Contains fields appropriate to the key level. There is one level total format per level break per report function (PRTOBJ or PRTFIL). By default, this format is printed after the last detail record or subtotal within a level break. If the access path contains only one key field, this format is omitted.
- The Final Total Forma—Contains totals of previous total levels. By default, this format contains only the constant Final Totals; user fields must be added for totaling. By default, this format is printed after the last level total format.
- The End of Report Format (report footer) at the end of the report—Is defined by a Define Report Format (DFNRPTFMT) function type. There is one End of Report format per PRTFIL. This format is printed after all other formats in the report. This format is not available for PRTOBJ.

These format types are present on PRTFIL and PRTOBJ functions.

The following example shows the general appearance of a report design, including report design format types and page headings.



You use the Display Report Formats panel to view which formats are present in the report device design. This is where you determine the available formats, if they are printed, the positioning and overflow treatments, and zoom into the format details. To access the Display Report Formats panel, press F17 from the report device design.

Automatic Choice of Report Formats

A single detail format is automatically defined containing all the fields from the underlying access paths.

Additional header and total formats are automatically defined for each key level present in the access path as follows:

- If the access path is not unique, header and total formats are defined for each key level present.
- If the access path is unique, header and total formats are defined for each key level present except for the lowest.

Report Header, Top of Page, First Page, and End of Report formats are provided automatically as follows:

- A First Page format is defined for both PRTOBJ and PRTFIL functions.
- A Report Header, Top of Page, and End of Report format are defined only for PRTFIL functions.

Formats are ordered relative to each other as follows:

- Formats appear in hierarchical order of level: overall headings, level break headings, detail record, level totals, and overall totals.
- Level headings appear in key order, as specified by the access path on which the report design is based.
- Each format starts on a new line. A default number of spaces are used between each format.
- Formats are indented.

```
HDR  Report headings
TOP  Topof page headings
1PG  First page headings

xHD  Level n headings
 . .
 xHD  Level 2 Headings
  xHD  Level 1 headings
   RCD  Detail format
  xTL  Level 2 totals
 xTL  Level 2 totals
 . .
nTL  Level n totals
. .
ZTL  Final report
EOR  End of report
```

Automatic Choice of Report Fields

Fields within a format are laid out left to right across the page in the order in which they appear in the access path. All fields from the access path are available on all formats but can be hidden or dropped by default.

Fields will, by default, be present or dropped at a given format level according to the following rules:

- Key fields are present on their respective heading and total formats and all lower level formats.
- Non-key fields that are virtual fields are present on a format if all the fields resulting from the resolution of the virtual field's defining relation are also present on the format. That is, if the necessary key fields to retrieve the virtual field are also present. Any such virtual fields are also present on all lower level formats.
- Key fields that are virtual fields are present together with their associated real fields if the print function is based on a Query access path. That is, the real fields resulting from the resolution of the virtual field's defining relation. These fields are also present on all lower formats.
- Non-key fields that are neither virtual fields nor fields associated with a virtual key are present only on the detail record.

Fields will, by default, be hidden or shown at a given format level according to the following rules:

- Key fields (including virtual keys) are, by default, hidden except on the respective heading and total formats which they control.
- Non-key fields that are virtual fields are shown on the format containing their controlling key fields, if any. If a virtual field is present on a higher-level format then, by default, it is hidden on the detail level format.
- Non-key fields associated with a virtual key are shown on the format containing their associated virtual keys. They are hidden on the detail format.
- Non-key fields which are neither virtual fields nor fields associated with a virtual key are, by default, shown on the detail format.

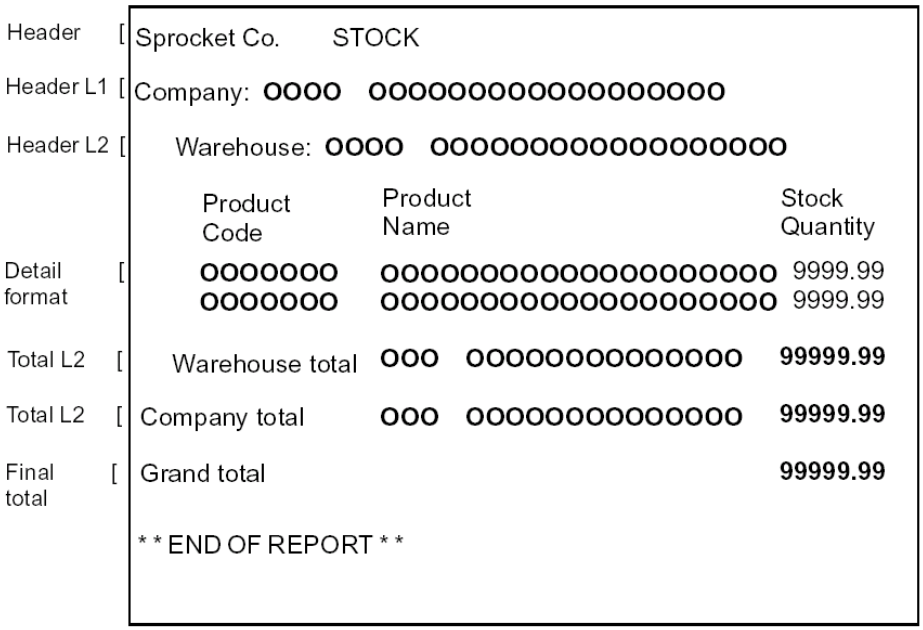
Access Path Entries	KHD	RCD	KTL
K Key field (real)	O	H	O
K Key field(Virtual)	O	H	O
Virtual field	O	H	O
Fld associated with virtual key	O	H	O
Detail field	–	O	–

Access Path Entries	KHD	RCD	KTL
Key O: Present and shown by default H: Present but hidden by default –: Dropped by default			

Field text is obtained as follows:

- If a field is present on a detail record, the Column Heading text is used as the text heading for the field
- If a field is present on a heading or total format, the Before text is used as the label for the field.

The following example shows a report device design made up of eight formats:



Defining Report Designs

There are two steps involved in defining a report with CA 2E.

- Specification of the individual report design: a default report design is provided automatically for each print function such as each PRTOBJ or PRTFIL, that you define. You can then modify the design.
- Inclusion of the individual report designs within an overall device structure. You specify how this is done explicitly, by connecting the functions together and specifying the parameters to pass between the functions.

You can modify default report layout at both the format and field level. Each of these steps is identified as follows.

You can modify the device design at the format level to:

- Suppress (by either dropping or hiding) formats
- Modify the spacing between formats
- Specify whether the format is to be reprinted on overflow
- Modify the format indentation

You can modify the device design at the field level to:

- Rearrange the order of fields
- Drop fields or field text
- Add extra fields or text
- Modify field text

When adding extra fields you can specify totaling between formats.

Suppressing Formats

By default, a report design has all of the appropriate formats present. You can suppress a report format in one of two ways:

- Hiding
- Dropping

Hiding

Hiding a report format causes the printing of the format to be suppressed. The format is still logically present. This means that any fields belonging to the format are available for processing, and any level break processing takes place.

Dropping

Dropping a report format causes the format to be omitted completely. Any fields belonging to the format are no longer available for processing. Level break processing, such as printing embedded PRTOBJ functions still takes place. Generally, it is better to drop a format rather than hide it, if the format is not required on the report.

Note: Function fields and constants cannot be copied between formats.

Format		Drop Format Allow	Drop Format Default	Hide Format Allow	Hide Format Default
Report Heading	HDR	N	-	N	-
Top of page heading	TOP	Y	N	Y	N
First page heading	1PG	Y	N	Y	N
Level heading	KHD	Y	N	Y	N
Detail format	RCD	N	-	Y	N
Level total	KTL	Y	N	Y	N
Final total	ZTL	Y	N	Y	N
End of report	EOR	N	-	N	-

Modifying Spacing Between Formats

CA 2E lets you specify, as part of your device design, any page overflow handling.

Specifying Print on Overflow

You can specify print overflow at the format level on the Display Report Format Details panel including

- How many lines to skip before the format is printed, or a start line
- Whether the format is to start on a new page
- Whether the format is to be reprinted on overflow

The following table displays the print control defaults.

Format		Start Line		Space Before		Start New Page		Reprint Overflow	
		Alw	Dft	Alw	Dft	Alw	Dft	Alw	Dft
Report heading	HDR	Y	1	Y	-	R	Y	R	Y
Top of page heading	TOP	Y	-	Y	1	N	-	N	-
First page	1PG	Y	-	Y	1	N	-	N	-
Level heading	KHD	Y	-	Y	1	Y	N	Y	N
Detail	RCD	Y	-	Y	1	Y	N	N	-
Level total	KTL	Y	-	Y	1	Y	N	N	-

Note: R = Required

Changing Indentation

Report design formats can be indented. Indentation controls the position on the printed report of the starting point of a format. You can control indentation at two levels:

- Individual format level: Specify an indentation on each format.
- Function level: Specify an indentation for a PRTOBJ function, then all formats belonging to the function are indented by the specified amount. For example, the baseline of the function is indented.

There are two types of indentation specifications:

Absolute

The indentation of a format is relative to the left-hand margin of the whole report and is unaffected by changes to the indentation of other formats around it. Absolute indentation applies to formats only.

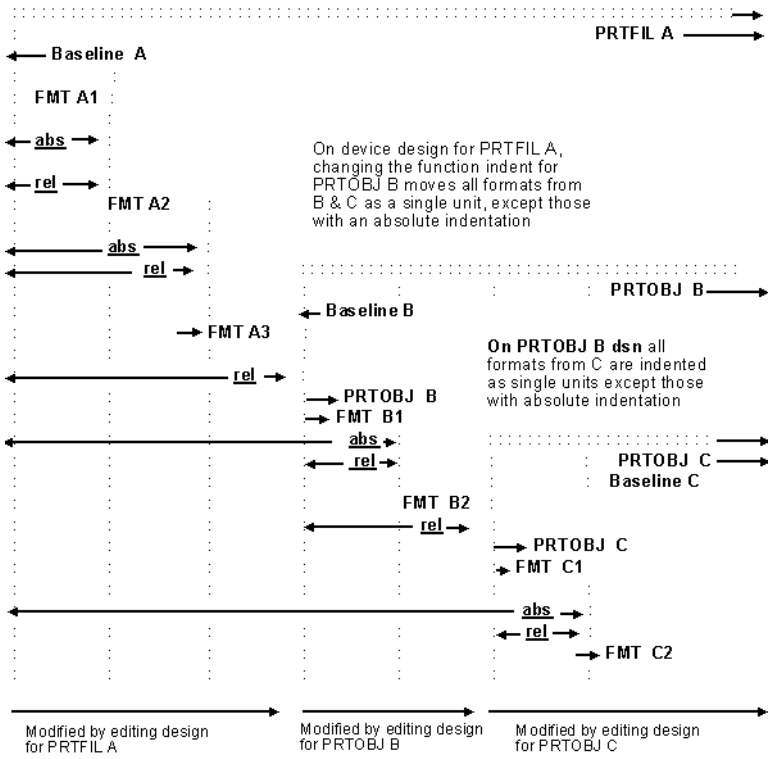
Relative

The indentation is relative to the baseline of the report function that you are currently editing. Function indents are always relative; format indents can be relative or absolute.

Modifying Indentation

To modify indentation for formats, use the Edit Report Format Details panel. To modify indentation for embedded PRTOBJ functions, use the Edit Function Indent panel. You can get to both panels through the Display Report Formats panel by pressing F17 on the embedded PRTOBJ function. You can only modify the format indentation for the formats that belong to the function whose design you are editing. The format indentation of formats belonging to embedded PRTOBJ functions can only be altered by editing the design for the individual PRTOBJ function.

The following is an example of the indentation formats.



Defining the Overall Report Structure

Individual report segments are combined into an actual report by means of a device structure. A device structure specifies which PRTOBJ functions are to be embedded in the report and at which points:

- You can link one or more print object functions before or after all report format types except the header, footer, and top-of-page formats.
- A call to each embedded PRTOBJ function is added to the action diagram of the embedded function.

The following table shows the allowed points for embedded PRTOBJ calls.

Format		Embed Before	Embed After	Indentation	
				Allow	Default
Report Heading	HDR	-	-	-	-
Top of page heading	TOP	-	-	-	-
First page heading	1PG	Y	Y	Y	0
Level heading	KHD	Y	Y	Y	+3
Detail format	RCD	Y	Y	Y	+3
Level total	KTL	Y	Y	Y	-3
Final total	ZTL	Y	Y	Y	0
End of report format	EOR	-	-	-	-

Modifying the Overall Report Structure

Individual report segments (formats) can be combined into an actual report using a device structure. This is only required if you have a combination of a PRTFIL with one or more embedded PRTOBJs. A device structure specifies which PRTOBJ functions are called in the print file of another PRTOBJ and at which points.

- Link one or more PRTOBJ functions before or after all report format types except the header, footer, and top-of-page formats.
- A call to each embedded PRTOBJ function is added to the action diagram of the function. This must be a PRTFIL or another PRTOBJ.

Defining Print Objects Within Report Structure

The overall structure of report designs can be edited using the Edit Device Structure program. This structure is accessible from the Edit Functions and Display All Functions panels by typing the line command **T** next to the calling PRTFIL or PRTOBJ. You can use the structure editor to:

- Link in additional PRTOBJ functions before or after each of the report formats
- Change, delete, move, or copy PRTOBJ calls within a report structure

Note: If parameters are to be passed between print functions, these parameters must be declared through the action diagram of the embedding function. For example, you must go to each function call in the appropriate action diagram.

Using Line Selection Options

When editing the overall structure of the report designs on the Edit Device Structure panel, use the following line selection options.

Value	Description
IA	Insert After
IB	Insert Before
D	Delete
Z	Zoom into the structure of embedded PRTOBJ
C	Copy: B-Before, A-After
M	Move: B-Before, A-After

Linking Print Functions

Embedding PRTOBJ functions is subject to the following rules:

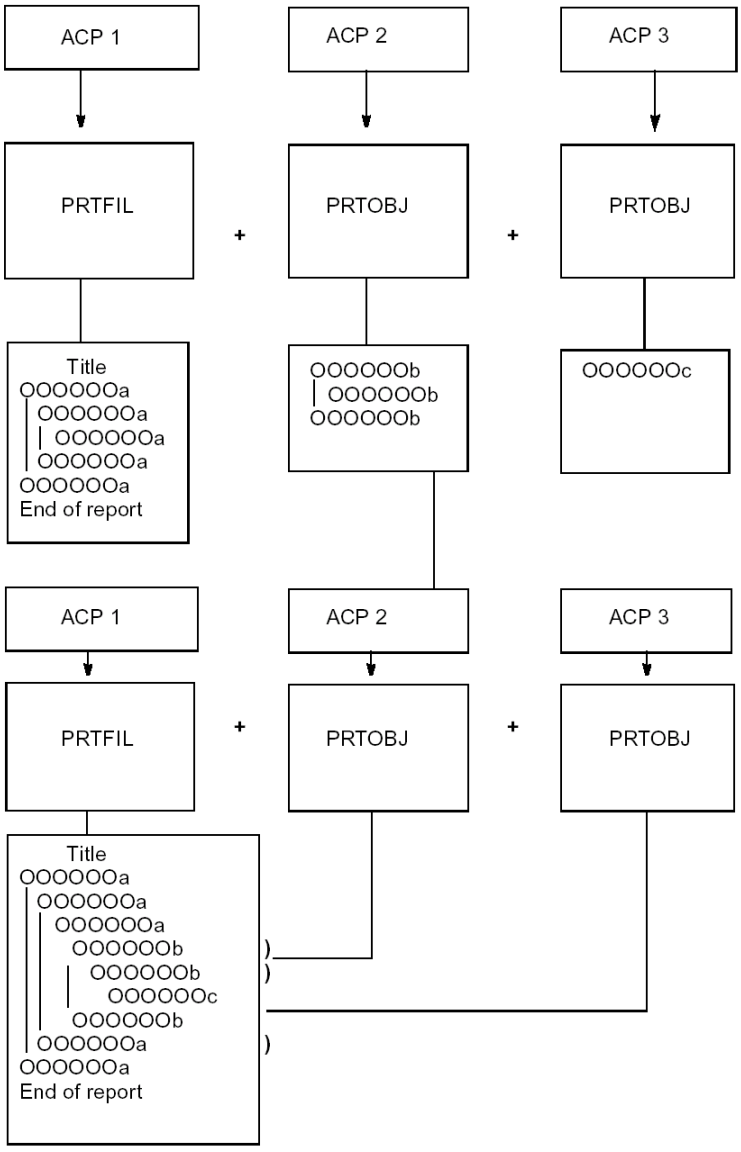
- You can insert PRTOBJ functions before or after some of the formats in a report function. More than one PRTOBJ function can be inserted at each point.
- Indentation of embedded PRTOBJ report segments (function indentation) is relative with respect to the baseline of the function. However, the indentation of formats within the PRTOBJ function may be absolute with respect to the left-hand margin of the complete report. The function indentation is a property of the calling function. Thus, the same PRTOBJ function can be used in two different PRTFIL functions with a different function indentation in each.
- You can embed PRTOBJ functions within other PRTOBJ functions.
- A PRTOBJ function can be used in more than one other PRTFIL or PRTOBJ function. A given PRTOBJ function can appear more than once in a given PRTFIL function but it cannot be called recursively. For example, a PRTOBJ function must not call itself either directly or indirectly.

Zooming into Embedded Print Objects

When editing report design structures, such as linking one or more functions or inserting subsidiary PRTOBJ functions, Zoom into the structure of the embedded PRTOBJ. At the Edit Device Structure panel, type Z next to the selected function to zoom into the structure.

Note: When editing the device design of a PRTFIL or PRTOBJ that contains an embedded PRTOBJ, the design of the combined functions displays. Moreover, you can only change the formats and entries of the function itself and not those associated with the embedded functions.

The following example shows an outline of linking report functions.


ACP 1
ACP 2
ACP 3
PRTFIL
+
PRTOBJ
+
PRTOBJ
Title
OOOOOOa
OOOOOOa
OOOOOOa
OOOOOOa
OOOOOOa
End of report
OOOOOOb
OOOOOOb
OOOOOOb
OOOOOOc
ACP 1
ACP 2
ACP 3
PRTFIL
+
PRTOBJ
+
PRTOBJ
Title
OOOOOOa
OOOOOOa
OOOOOOa
OOOOOOb
OOOOOOb
OOOOOOc
OOOOOOb
OOOOOOa
OOOOOOa
End of report

Using Function Fields on Report Design

You will want to accumulate the results of calculations upwards through each level break. You can use function fields to do this.

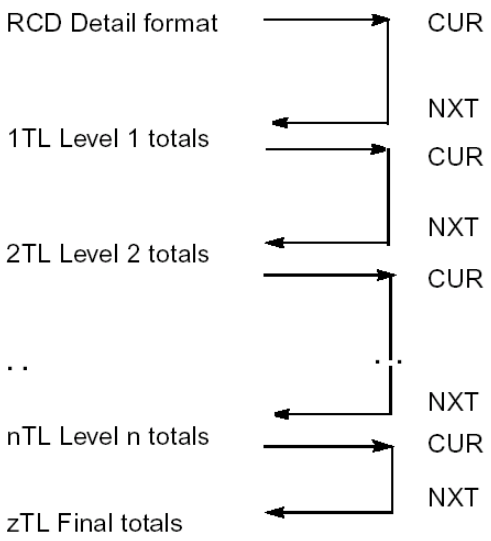
Function field types include four standard types that provide predefined functions to carry out calculations between two adjacent levels of a report. The predefined types are:

- **SUM**—Accumulates a value from the current level into a total field on the next level.
- **CNT**—Accumulates a count of the number of instances of a field on the current level (that is, records containing the field) into a field on the next level.
- **MAX**—Places the largest value of a field on the current level into a field on the next level.
- **MIN**—Places the smallest value of a field on the current level into a field on the next level.

You can specify that the input to a calculation (for example, an accumulation) is the respective field from the previous level. You do this by means of the CUR and NXT contexts. If the function field definition is based on the field, CA 2E defaults the context automatically.

For more information on CUR and NXT contexts, see Understanding Contexts in the chapter "Modifying Action Diagrams."

The following example shows the use of CUR and NXT contexts in Reports.



Report Design Example

To design reports with CA 2E, consider the structure of the data, and choose the appropriate function combinations that map to the structure. The following two-part example illustrates this by showing:

- A simple PRTFIL report design. (Example 1)
- The same PRTFIL report design with embedded PRTOBJ functions (Example 2)

Each part of the example shows the following:

- Relations and underlying structure of the data
- Resulting access path entries
- Resulting report formats
- Fields present on the report formats
- Resulting report layout

Example 1: Simple Report Design

Relations

Consider the following relations to model customer information by geographical region:

```
FIL   Country   REF   Known by   FLD   Country code     CDE
FIL   Country   REF   Has        FLD   Country name     TXT

FIL   Area      REF   Owned by   FIL   Country          REF
                                 VRT   Country name     TXT
FIL   Area      REF   Known by   FLD   Area code        CDE
FIL   Area      REF   Has        FLD   Area name        TXT

FIL   Customer  REF   Owned by   FIL   Area             REF
                                 VRT   Country name     TXT
                                 VRT   Area name        TXT
FIL   Customer  REF   Known by   FLD   Customer code    CDE
FIL   Customer  REF   Has        FLD   Customer name    TXT
FIL   Customer  REF   Has        FLD   No of machines   NBR
```

If you wanted to produce a report of customers by geographical region, you would need the following information:

Access Path Entries

A Retrieval access path built over the Customer file might contain the following entries:

Note: If you use a RSQ or QRY access path, you would specify the keys yourself.

Access Path Entries	Type	Key
Country code	A	K1
Country name	V	K2
Area code	A	K3
Area name	V	
Customer code	A	
Customer name	A	
No of machines	A	

Default Report Formats

A report design built over the access path shown in the preceding topic would contain the following report design formats:

Format	Type	PRTFIL	Fields
STD report header	HDR	Y	DFNRPTFMT
Top of page	TOP	Y	User
First page format	1PG	Y	User
Country code	1HD	Y	Country
Area code	2HD	Y	Area
Detail line	RCD	Y	Customer
Area code	3TL	Y	Area
Country code	4TL	Y	Country
Final totals	ZTL	Y	User
End of report	FTR	Y	DFNRPTFMT

Report Design Fields by Format

The device design formats would, by default, contain the following fields:

Access Path Entries			Report Formats				
			1HD	2HD	RCD	3TL	4TL
K1	Country code	A	O	H	H	H	O
K2	Country name	V	O	H	H	H	O
K3	Area code	A	-	O	H	O	-
	Area name	V	-	O	H	O	-
	Customer code	A	-	-	O	-	-
	Customer name	A	-	-	O	-	-
	No of machines	A	-	-	O	-	-
Key	O: Present and shown by default H: Present but hidden by default - : Dropped by default						

Consequently, the default report structure would be:

```
HDR Page headings
 1PG First page
   1HD Country code header
     2HD Area code header
      RCD Customer details
     3TL Area code totals
   4TL Country code totals
  ZTL Final totals
FTR End of report
```

The default report design might then appear as follows:

```
                                                    *USER  *DATE *TIME
HDR  [                          Print Customers
TOP  [
1PG  [
1HD  [   Country code OOO  Country name OOOOOOOOOOOO
2HD  [      Area code OOOO  Area name OOOOOOOOOOO
                Customer    Customer name                 No of
RCD  [          code                                      machines
                OOOOOO      OOOOOOOOOOOOOOOOO             OOOOO
                OOOOOO      OOOOOOOOOOOOOOOOO             OOOOO
                OOOOOO      OOOOOOOOOOOOOOOOO             OOOOO
3TL  [      Area code OOOO  Area name OOOOOOOOOOO
4TL  [   Country code OOO  Country name OOOOOOOOOOOO
ZTL  [   Final totals
FTR  [   ** END OF REPORT **
```

Function Fields

You can also add to the report device design a total count of the number of machines on each format. To do this, you use the function fields as follows:

1. Define a function field Total Number of Machines of the type SUM, based on the No of Machines field.
2. Add this field to the Area, Country, and Final Total formats accepting the default parameters.

To add the function fields to accumulate the number of customers you can:

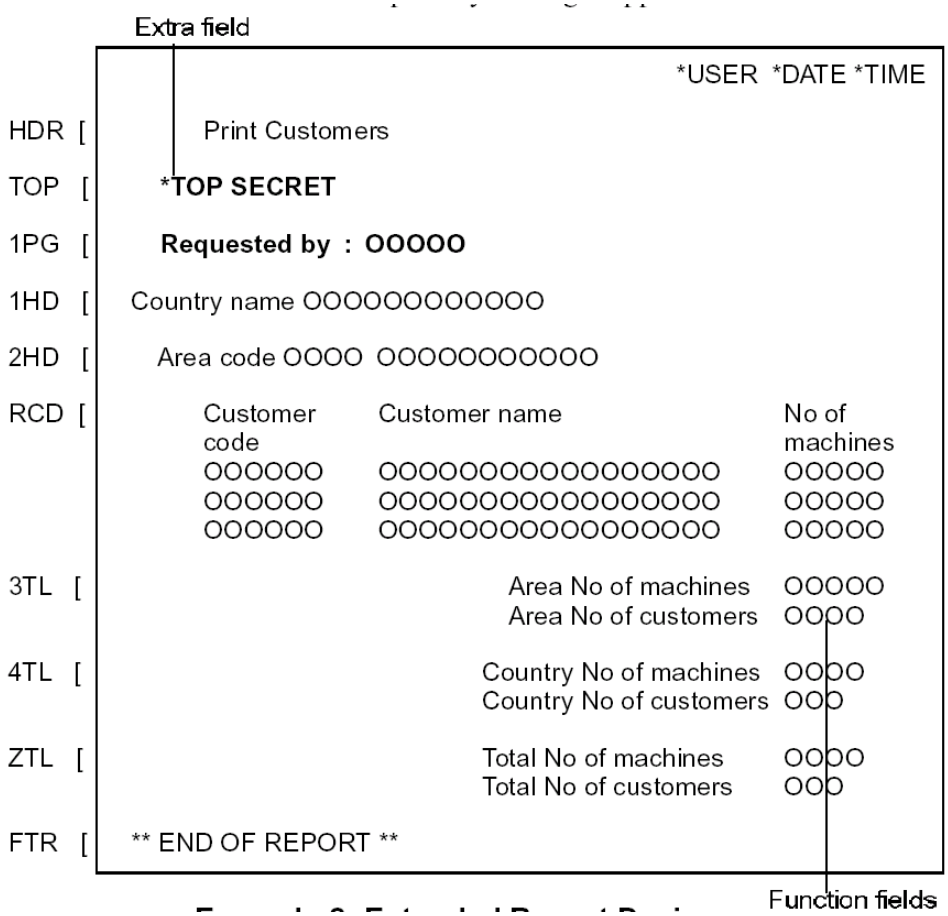
1. Define a CNT function field, Count No. of Customers, with Customer Code field as the input parameter.
2. Add the CNT field to the Area Total Format.
3. Define a SUM function field, Total of customers, based on the Count No. of Customers field.
4. Add the SUM field to the Country and Final Total formats, accepting the default parameters.

Modified Report Layout

You can modify the default report layout as follows:

- To add explanatory information on the page headings (for example, *TOP SECRET)
- To suppress certain fields, or field text (for example, the Country Code field and the text on the Area Name field) and the fields displayed by default on the total formats.

The modified report layout might appear as follows:



Example 2: Extended Report Design

You can extend the example report design shown earlier by embedding additional PRTOBJ functions. For example, you could have additional entities and PRTOBJ functions based on them as follows:

- County (within Area)
- Distributor (within Country)
- Address (by Customer)
- Orders (for Customer)

County Report Segment

If Area is divided into County as shown by the following relations:

```
FIL   Country  REF  Known by   FLD   Country code   CDE
FIL   Country  REF  Refers to  FIL   Area           REF
FIL   Country  REF  Has        FLD   Country name   TXT
```

Then to provide a sublisting of the counties for each area, you need a RSQ access path with the following entries:

Access Path Entries	Type	Key
County code	A	K3
Country code	A	K1
Area code	A	K2
County name	A	

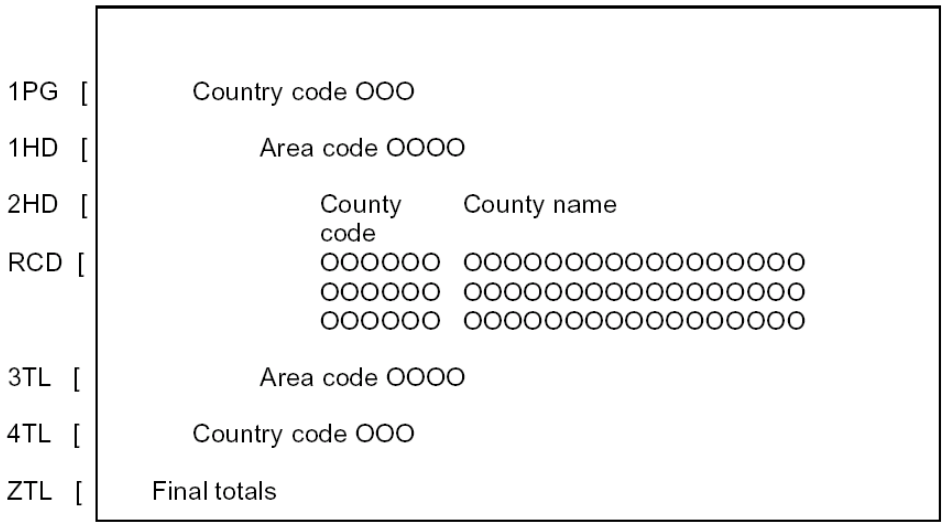
Note: This access path should be specified so that it has a unique key order.

A Print Counties in Area PRTOBJ function would be based on this access path. Country Code and Area Code would be made restrictor parameters so that only counties for a given area would be printed:

PRTOBJ Function	ACP	Keys on Access Path	USG
Counties in area	RSQ	1. Country code	RST
		2. Area code	RST
		3. County code	

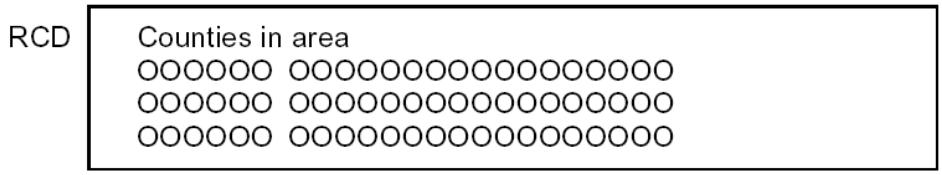
County Default Report Design

This creates a default layout for the report design as follows:



County Modified Report Design

By dropping the formats, the suppression of headings would appear as follows:



Distributor Report Segment

If in each country there are distributors as defined by the following relation:

Distributor Relations

```
IL   Distributor  REF  Owned by      FIL   Country           REF
FIL   Distributor  REF  Known by      FLD   Distributor code  CDE
FIL   Distributor  REF  Has           FLD   Distributor name  TXT

FIL   Area         REF  Refers to     FIL   Distributor       REF

  For: Default                      Sharing: *ALL
```

To provide sublistings of the following, we need a PRTOBJ on Distributor (Distributors in Country) and a PRTOBJ on Area (Default Area Distributor), both based on the Retrieval access paths of the respective files.

Examples:

- The distributors for each country
- The default distributor's details for each area

Distributor Access Path Entries

Access Path Entries	Type	Key
Country code	A	K1
Distributor code	A	K2
Distributor name	A	

Area Access Path Entries

Access Path Entries	Type	Key
Country code	A	K1
Area code	A	K2
Distributor code	A	
Area name	A	

Country code would be made a restrictor parameter on the Distributors in Country function so that only distributors in the given country would appear. Both Country code and Area Code would be made restrictors on the Default Area Distributor function so that only the details for the specified distributor would print.

Distributor PRTOBJ Functions

PRTOBJ Function	ACP	Keys on Access Path	USG
Distributors in country (Distributor file)	RTV	1. Country code 2. Distributor code	RST
Dft area distributor (Area file)	RTV	1. Country code 2. Area code	RST RST

Distributor Modified Report Design

This function gives a layout, after modification, for the Distributors in Country report design as follows:

```
RCD     Distributors
              OOOO OOOOOOOOOOOOOO
              OOOO~OOOOOOOOOOOOOO
              OOOO OOOOOOOOOOOOOO
```

A layout, after modification, for the Default Area Distributor report design as follows:

```
RCD     Distributors
        Area Default Distributor OOOOOOOOOOOOO
        Area Default Distributor OOOOOOOOOOOOO
        Area Default Distributor OOOOOOOOOOOOO
```

Note: Only one distributor will be printed as the function is fully restricted. The device design shows three lines for the detail line format.

Address Report Segment

If each Customer has an Address as defined by the following relations:

```
FIL   Address   REF   Owned by  FIL   Customer       REF
FIL   Address   REF   Has       FLD   Address line 1 TXT
FIL   Address   REF   Has       FLD   Address line 2 TXT
FIL   Address   REF   Has       FLD   Post code      TXT
```

Then to provide a sublisting of the address for each customer, you would need a RTV access path with the following entries:

Address RTV Access Path Entries

Access Path Entries	Type	Key
Country code	A	K1
Area code	A	K2
Customer code	A	K3
Address line 1	A	
Address line 2	A	
Post code	A	

The Print Customer Address print function would have Country Code, Area Code, and Customer Code as restrictor parameters so that only the address for the given customer prints.

Address PRTOBJ Functions

PRTOBJ Function	ACP	Keys on Access Path	USG
Customer address	RTV	1. Country code	RST
		2. Area code	RST
		3. Customer code	RST

Address Modified Report Design

This gives a layout, after modification, for the Customer Address report design as follows:

```
RCD       Customer address   : OOOOOOOOOOOOOOO
                               OOOOOOOOOOOOOOO
          Post code . . . .  : OOOOOO
          Customer address   : OOOOOOOOOOOOOOO
                               OOOOOOOOOOOOOOO
          Post code . . . .  : OOOOOO
          Customer address   : OOOOOOOOOOOOOOO
                               OOOOOOOOOOOOOOO
          Post code . . . .  : OOOOOO
```

Note: Only one address is printed as the function is fully restricted. The device design shows three lines for the detail line format.

Order Report Segment

If, for each Customer, there can exist Orders as defined by the following relations:

```
FIL   Order   CPT   Known by   FLD   Order number   CDE
FIL   Order   CPT   Refers to  FIL   Customer       REF
FIL   Order   CPT   Has        FLD   Order date     DTE
FIL   Order   CPT   Has        FLD   Order value    VAL
```

To provide a sublisting of the orders for each customer, you would need an RSQ access path with the following entries:

Order RSQ Access Path Entries

Access Path Entries	Type	Key
Order number	A	K1
Country code	A	K2
Area code	A	K3
Customer code	A	K4
Order date	A	
Order value	A	

The Customer's Orders function would be restricted on Country Code, Area Code, and Customer Code.

Order PRTOBJ Functions

PRTOBJ Function	ACP	Keys on Access Path	USG
Customer's orders	RSQ	1. Country code	RST
		2. Area code	RST
		3. Customer code	RST
		4. Order date	

Order Function Fields

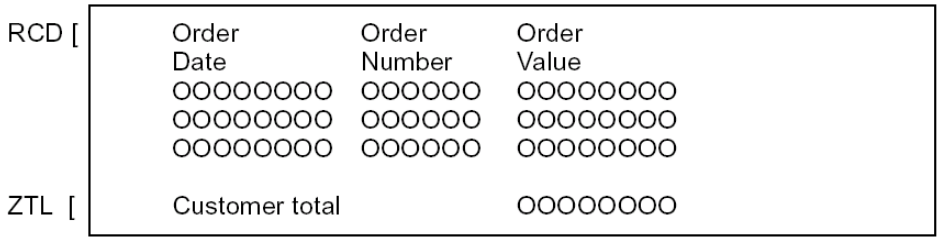
You make a further modification to the Print Customer Orders function to accumulate the total order value and print it. You can also return the total value so that it can be accumulated by area and country.

To do this:

1. Define a SUM function field, Total Order Value, based on the Order Value field.
2. Add this field to the Final Totals format, accepting the default parameters. All intermediate formats must be dropped.
3. Specify the Total Order Value as an output parameter for the PRTOBJ function, so that the values can be summed into the Area, Country, and Final totals of the PRTFIL function.
4. In the action diagram of the PRTOBJ function, move the calculated Total Order Value into the PAR context.

Order Modified Report Design

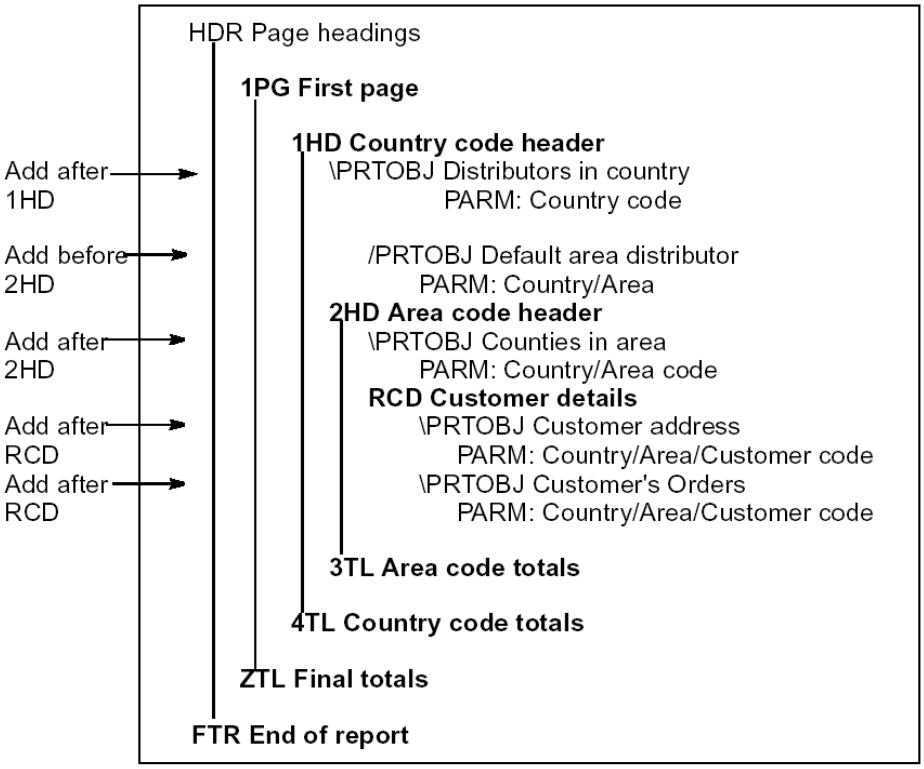
You would obtain a layout, after modification, for the Customer's Orders report design as follows:



Overall Device Structure

Having created all of the separate PRTOBJ functions as described previously, you would then insert them, using the device structure editor, into the basic default structure of the PRTFIL function. The following illustration shows the device structure editor with inserted PRTOBJs and the parameters specified for each PRTOBJ.

Note: In the actual device structure editor, parameters are not shown.



Parameters to PRTOBJ Functions

To supply parameters to the PRTOBJ functions called by the PRTFIL Print Customers, you must edit each function call in the PRTFIL action diagram. In most cases, the default parameters can be accepted.

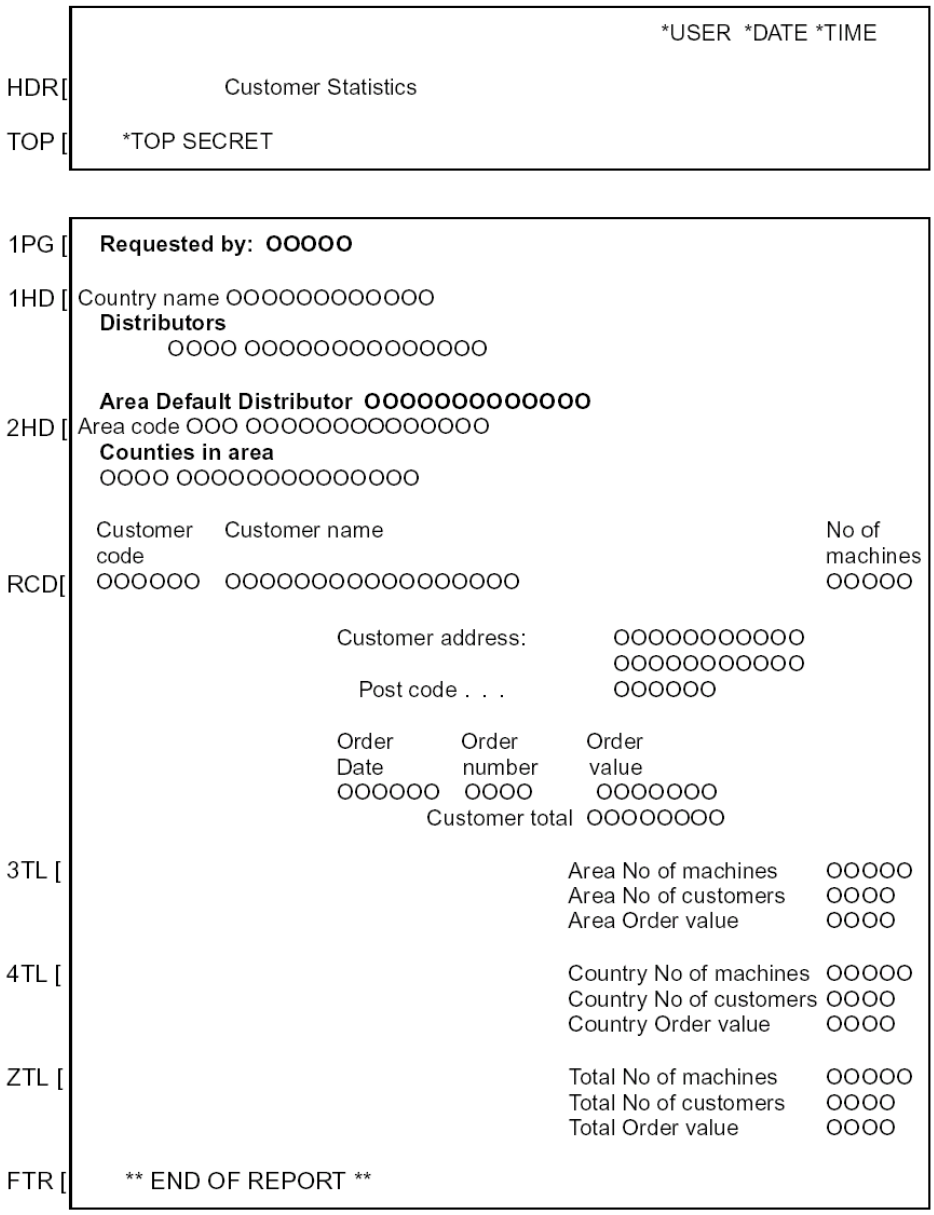
Function Fields

To accumulate the order value, do the following:

1. Define a USR field, Detail Line Order Value, based on the Order Value field on the Customer's Orders PRTOBJ function.
2. Attach this field to the Detail line format of the PRTFIL function, and hide it.
3. In the action diagram for the PRTFIL function, specify that the output parameter Total Order Value from the PRTOBJ function be moved into this USR field.
4. Define a SUM function field Accumulated Order Value based on the Order Value field.
5. Add this SUM field to the Area, Country, and Final Total formats, supplying appropriate parameters.

Overall Report Design

The separate functions shown previously would be combined to give the following overall result:



Device User Source

The device user source feature provides the ability to patch device designs of display and report functions in order to deploy operating system functionality that is not yet supported by CA 2E. For straightforward patches such as, inserting one or two data lines at the beginning of a generated source extent, you type the data lines in the device user source member that implements the patch. A set of special merger commands is provided for complex patches such as, applying a change to a specific location in the generated source based on a condition.

When to Use Device User Source

Device user source contains customized device language statements. Use device user source when:

- You require a feature that is not supported by CA 2E; for example, you need the following functions of the Advanced Printer Function (APF):
  - Print logos
  - Special symbols
  - Large characters
  - Bar codes
  - Bar charts
  - Vertical and horizontal lines that can be used to form boxes
- You require a facility introduced in the current i OS release that has not yet been implemented in CA 2E; for example, HTML support.
- The way an existing feature is implemented is not suitable for your use; for example, change appearance of an indicator to blinking or reverse image.
- You need to insert special anchors to support your pre-processor.

Understanding Device User Source

The term device user source refers to both:

- An EXCUSRSRC function that contains device language statements that can be applied to a device function to customize the associated device design
- The user-defined device language statements contained in the EXCUSRSRC function

User source attached to a device function is automatically reapplied each time the device function is regenerated.

Since DDS is the most commonly used device language, you can also use the more specific term DDS user source to refer to the EXCUSRSRC function and the user-defined DDS.

User-defined device code must use the same device language used by the device function to which it is applied. In addition, it is your responsibility to ensure that it follows the syntax rules of the device language.

The device EXCUSRSRC function may also contain special instructions called merger commands that specify how device user source is added to or substituted for automatically generated source for the device design.

The user-defined device source is stored as a separate member of the source file associated with the device language; for example, QDDSSRC in the generation library.

Attachment Levels

When you apply device user source to a device design property, it is said to be attached. The word property in this case refers to a specific instance of a device, screen, report, format, or entry. You can specify the attachment of device user source to the following types (levels) of device design properties:

- Device (device-level attachment)
- Screen/Report (screen- or report-level attachment)
- Format (format-level attachment)
- Entry (entry-level attachment)

Multiple device user source functions can be attached to a property. However, a specific device user function can be attached to a given property only once.

The term extent is used to refer to the set of source statements that describe a device design property in the generated source member.

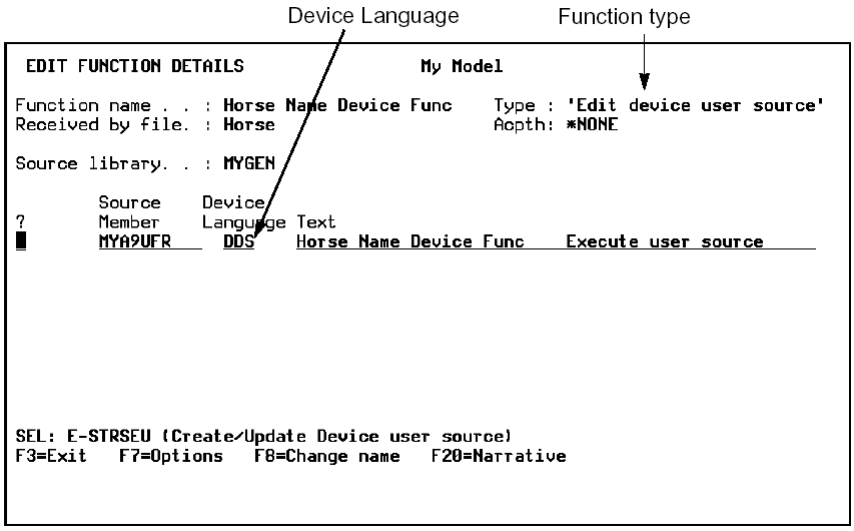
Special Field-Level Attachment

A field-level attachment provides an efficient way to apply a device entry change to all or most panels and reports that contain a particular field. In other words, when you attach device user source to a field, it is automatically attached to all derived device entries based on that field.

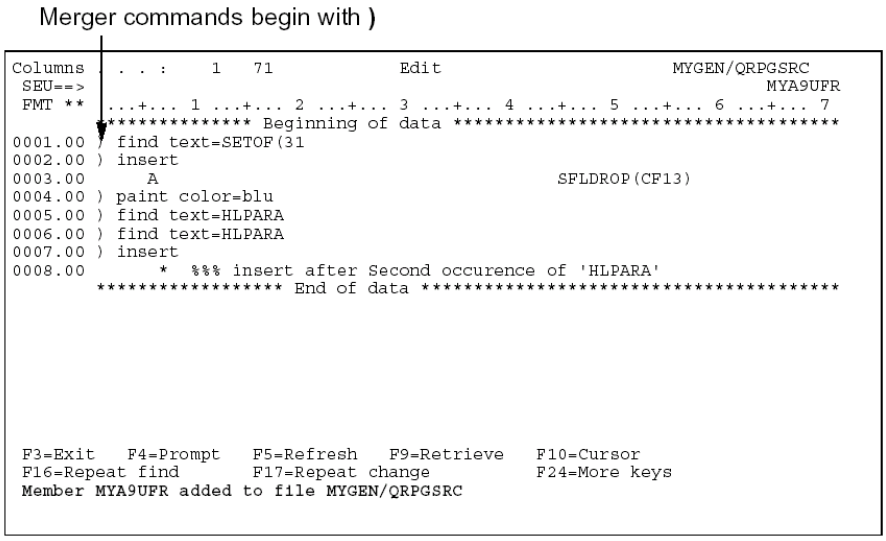
Defining a Device User Source Function

1. From the Edit Function panel, define an EXCUSRSRC function with access path *NONE.

   Zoom into the function. From the Edit Function Details panel, reset the device language, for example, to DDS.

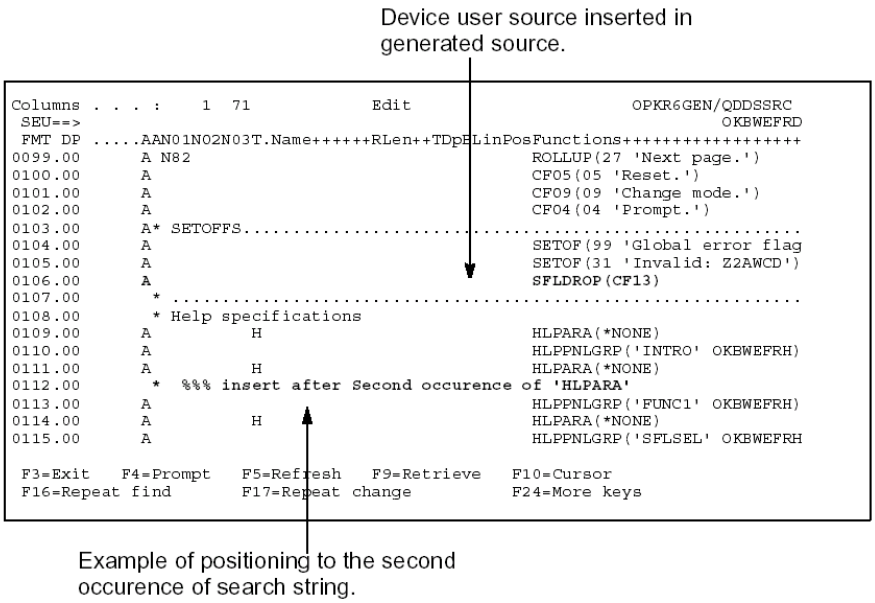


Type **E** against the file to enter DDS source or merger commands.



2. The commands beginning with) are special merger commands that are executed during source generation. Because merger commands do not follow DDS (or other device language) syntax, an error message displays and you need to set the 'Return to editing' option to N on the Exit Function Definition panel.

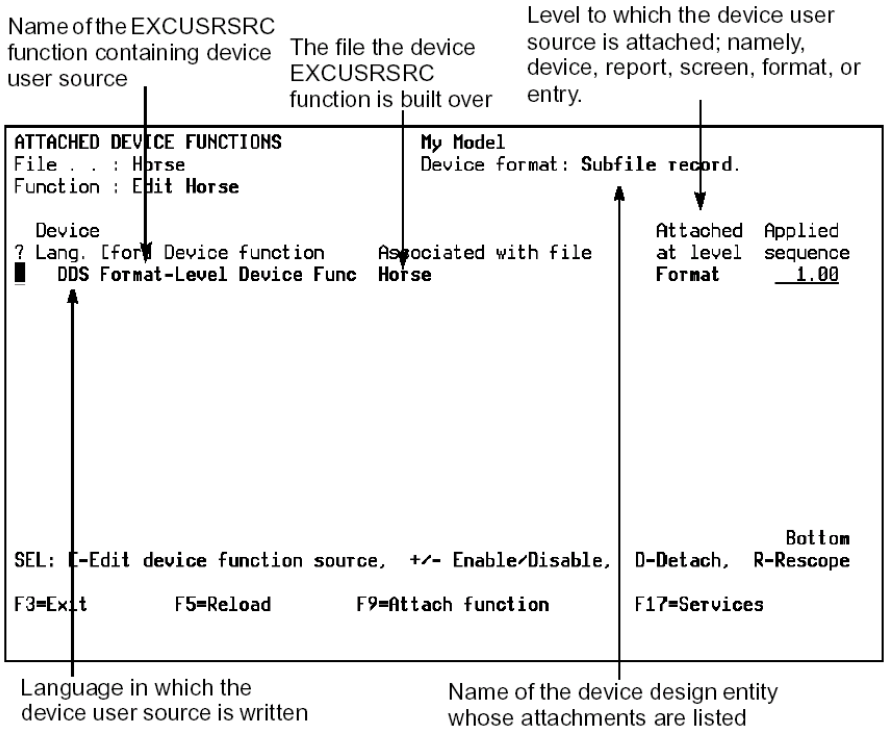
3. Save the source member. Device user source is merged when the function it is attached to is generated. Following is an example of device user source merged with generated source for the subfile control format of an Edit File function.



Attaching Device User Source to a Device Design

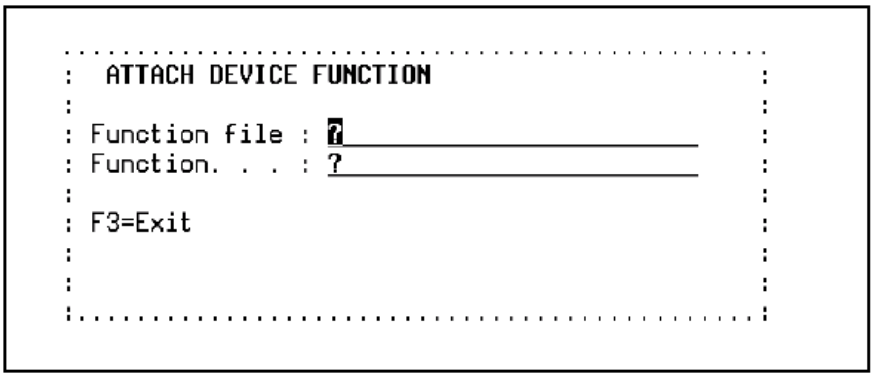
1. Go to the device design where you want to attach the device function. In this example, Edit Horse.
2. Do one of the following depending on the part of the device design where you want to attach the device user source.
   - For device-level, press F3.
   - For screen-/report-level, press F17.
   - For format-level, position the cursor on the selected format and press F5.
   - For entry-level, position the cursor on the selected entry and press Enter.

   Press F11 on the panel that appears to access the Attach Device Functions panel. In this example, we chose format level.

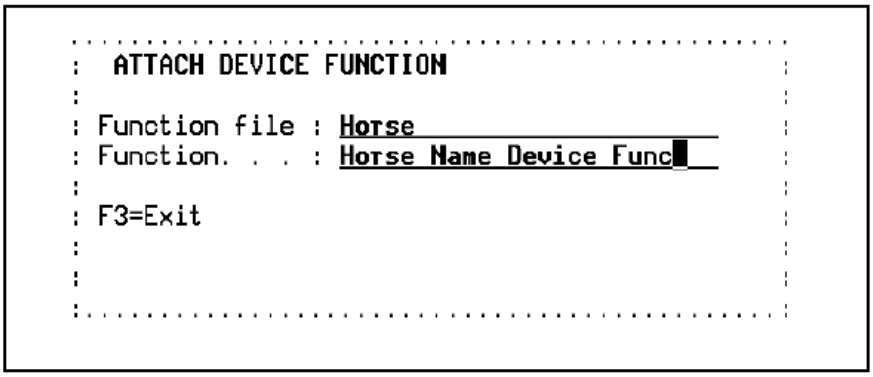


Note: A device user source function was previously attached to this format.

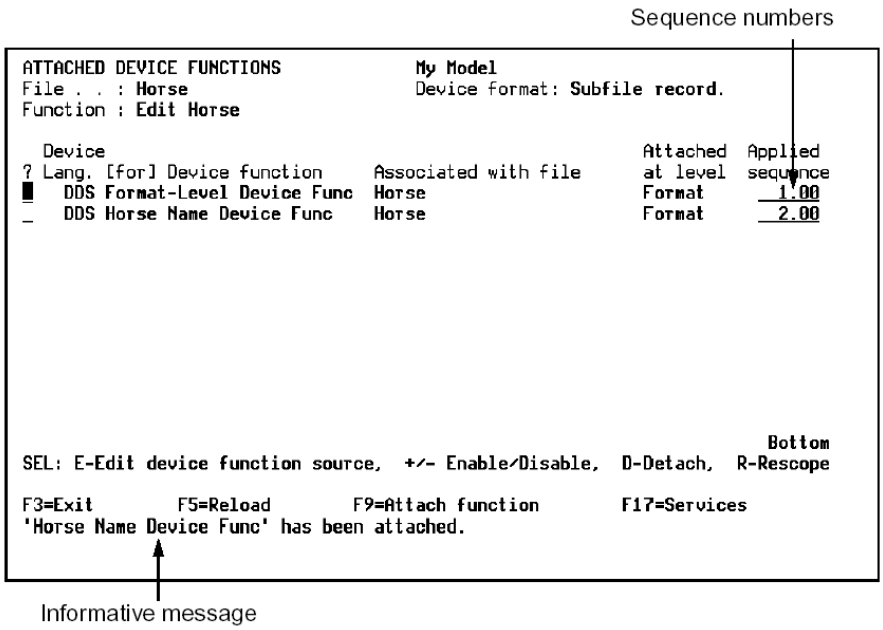
Press F9 to display the Attach Device Function window and attach the device user function just created.



Specify the device user function just created and press Enter.



The Attached Device Functions panel reappears showing both attached device user functions.



From this panel you can:

- Use the E option to edit device user source
- Use the D option to detach device user source
- Use the + and – options to temporarily disable the selected device user source

The sequence numbers specify the order in which the device user functions are attached to the device design. Sequence numbers are automatically assigned and can be changed. However, they must be unique for a level. The subfile is sorted by sequence number when the display is reloaded.

Sequence numbers:

- Range from 0.01 to 999.99
- Are incremented by 1.00 when automatically assigned

3. Press F3 to exit. The attached device user source is automatically applied to the subfile record of the Edit Horse device design whenever it is regenerated.

Entry-Level Device User Source

Device user source can be explicitly attached at the entry level or be implicitly attached through inheritance from the field level.

Explicitly Attaching Entry-Level Device User Source

From the Edit Screen Entry Details panel for the entry press F11 to display the Attached Device Functions panel. Press F9. The Attach Device Function panel displays where you can proceed as discussed in the section Attaching Device User Source to Device Design.

You can use the D option to detach the entry-level device user function that is explicitly attached. You can also use the + and – options to enable and disable the entry-level attachment.

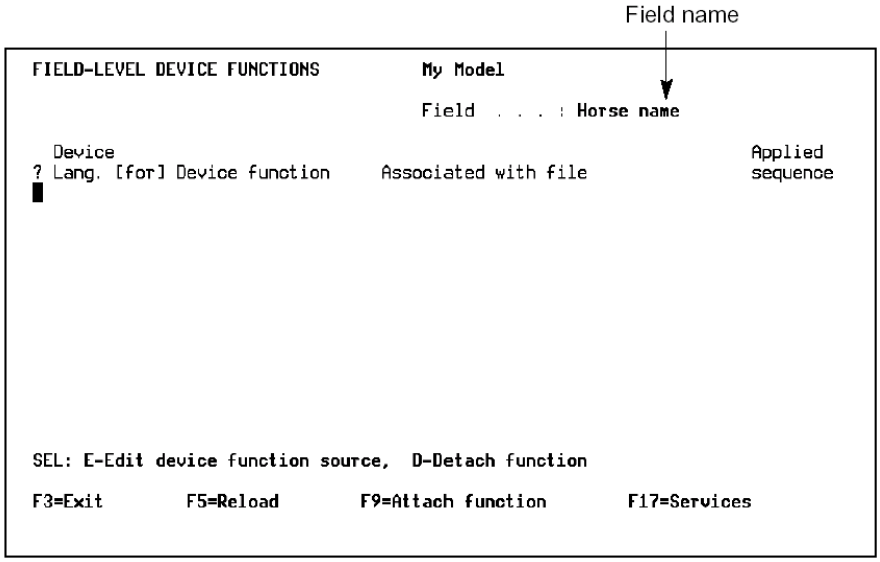
Note: When the sequence number for an explicitly attached entry-level device user function is assigned, numbers that are an even multiple of ten are skipped to avoid conflicts with field-level sequence numbers.

Attaching Device User Source to a Field

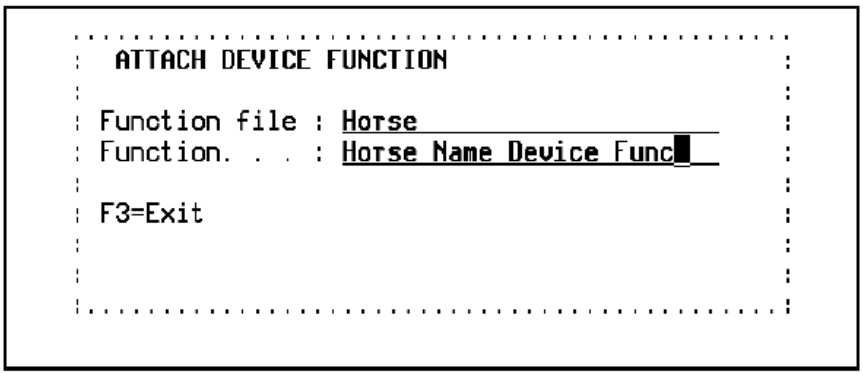
This example shows how to attach device user source to a field, which is then inherited by all device entries derived from this field.

1. Go to the Edit Field Details panel for the field to which the device function is to be attached. In this example, Edit Horse.

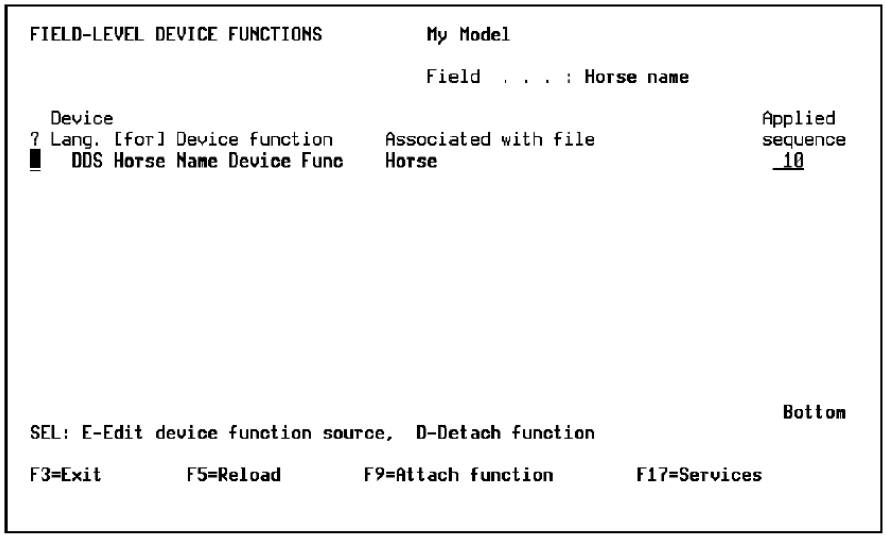
   Press F11 to display the Field-Level Device Functions panel. Use this panel to attach device user source to a field.



Press F9 to display the Attach Device Function window. Specify the device user function just created for the Horse name field and press Enter.



The Field-level Device Functions panel redisplays showing the attached device user function.



The sequence numbers specify the order in which the device user source is applied to the field. Field-level sequence numbers are:

- Integers from 1 to 999
- Incremented by 10 when automatically assigned

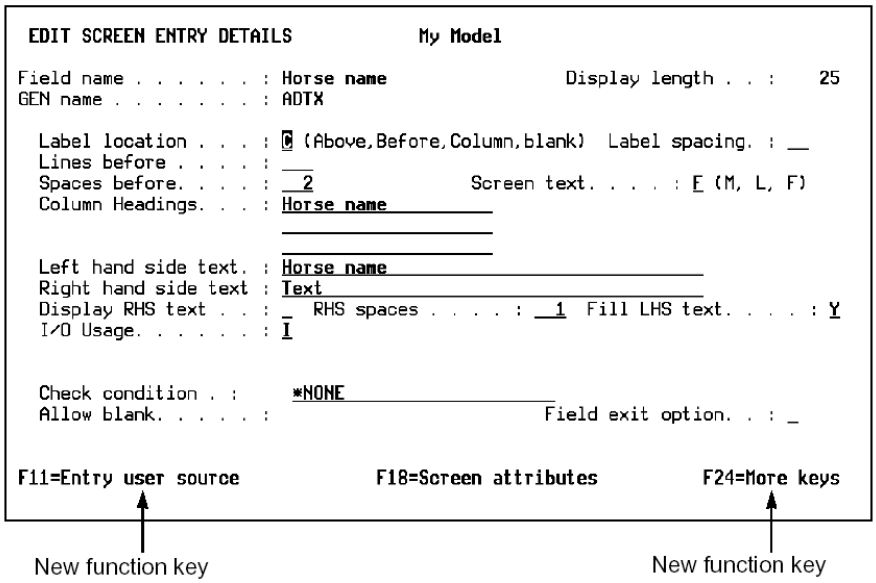
2. Press F3 to exit. The device user source is now implicitly attached to derived entries for the Horse name field on all device designs.

   Note: If a device user function is already attached at the entry level when you attach the same function at the field level, the field-level attachment for that entry is effectively ignored.

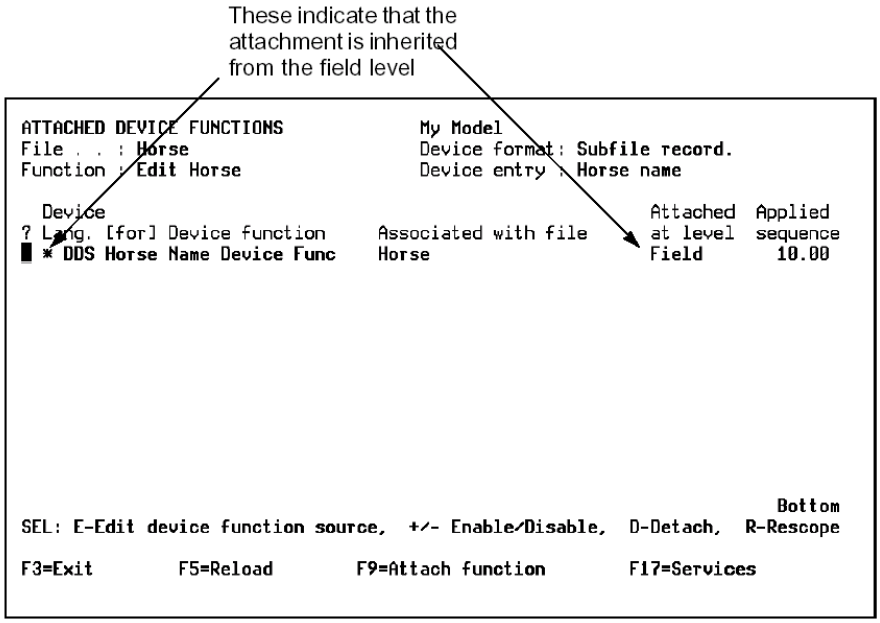
Working with Inherited Entry-Level Attachments

1. Go to a device design where the entry is used. In this example, Edit Horse.

   Position the cursor on the entry derived from the field where you attached the device user source; in this example, Horse name. Press Enter to display the Edit Screen Entry Details panel.



   Press F11 to display the Attached Device Functions panel for the Horse name entry. The device user source you attached to the Horse name field is shown in the list of device user functions attached to the Horse name entry.



Overriding an Inherited Entry-Level Attachment

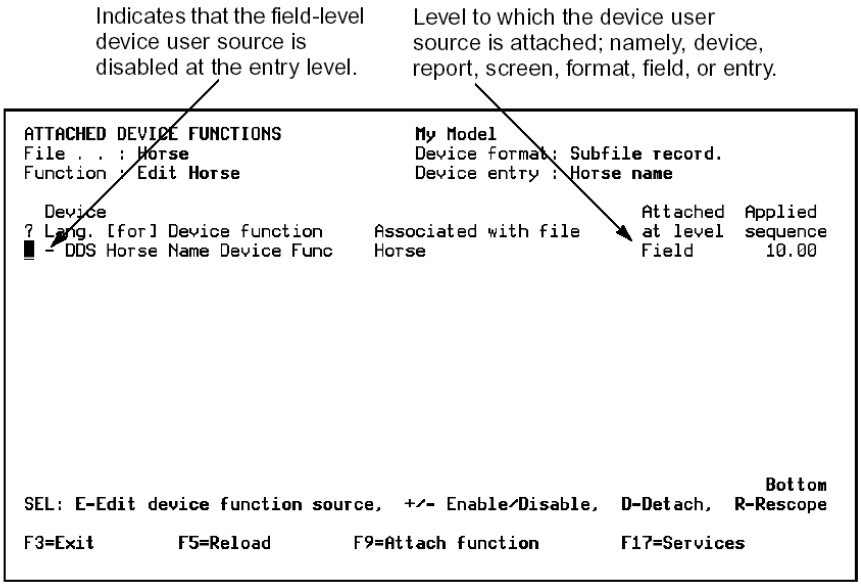
You can override an automatically attached device user source at the entry level in two ways:

- Disable the attached device user source at the entry level using the - option. This blocks the inheritance from the field level.
- Explicitly attach the device user source at the entry level using the R (rescope) option.

Note: You cannot use the D option on this panel to detach a device user function that was inherited from the field-level.

The + and - options are available for all device properties. You can use them to release or temporarily hold any explicitly attached device function. In addition, for the entry level you can use them to allow or prevent the inheritance of device user source from the field level.

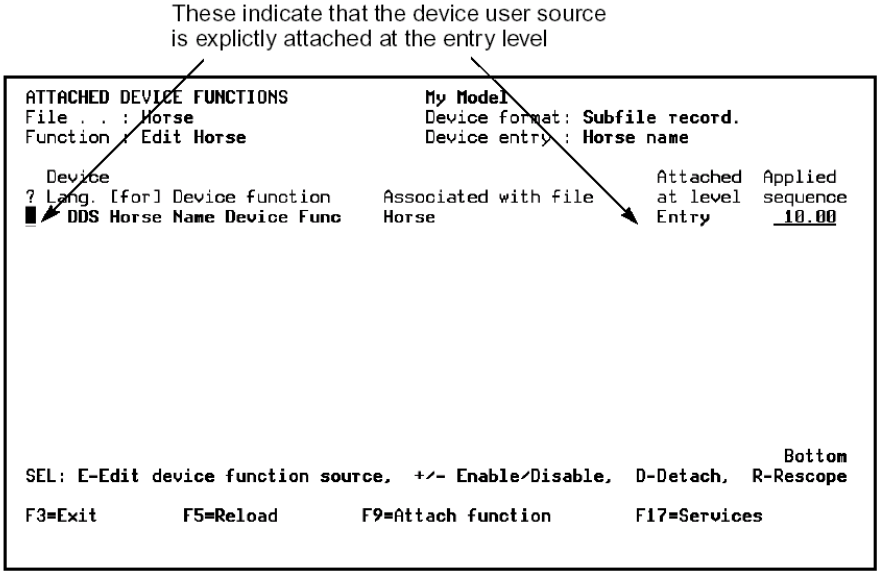
If you do not want the inherited device user function applied to this Horse name entry, type - against it and press Enter.



The device user function is no longer highlighted and the * is reset to - indicating that the function is disabled. When you generate the Edit Horse function, the device user source is not applied to the Horse name entry.

1. To re-enable the field-level device user source, type + against the function and press Enter. The * is redisplayed to indicate that the device user source is again inherited from the field level.
2. To make the entry-level user source independent of the field-level user source type **R** (Rescope) against it and press Enter. This explicitly attaches the same device user function to the entry, overriding the inheritance of the device user source attached at the field level.

   You can now detach the device user source at the field level without affecting the entry-level attachment.



   To reinstate the automatic field-level attachment, detach the entry-level attachment by typing D against it and pressing Enter.

Substitution Variables

The following substitution variables let you embed source generation information into Device User Source that is then resolved into the actual values in the final display file source.

Variable Name	Description
#*MBR	Display file source member name
#*FILE	Display file source file name

Variable Name	Description
#*LIB	Display file source file library name
#*TYPE	Display file source type (DDS)
#*ENT	Dependent on the attachment level as follows: Device: Display file name Screen/Report: blank (not supported) Format: Record format name Entry: Entry (field) name

Merger Commands for Device User Source

Device user source consists of DDS (or other device language) source statements and special merger commands specifying how to update the automatically generated source for the device design.

Each time a property of a device design is generated a special program, Device User Source Merger (the merger), is invoked to merge any device user source attached to the property. The merger interprets all encountered commands and updates the original generated source according to the requested actions.

Merger commands provide basic features available in classic line-editing word processors. The main concept of such tools is the *current line* a floating anchor around which the original text is updated. Two types of merger commands are required to complete similar tasks within a device user source function: one to position to the line in the source and another to edit the line.

Several device user source functions can be attached to a device property. When the first one is applied, the current line is the first line in the generated extent (default). As a result of the positioning instructions coded in the device user function, the location of the current line is changed. For subsequent device user source functions, the current line is not reset back to the beginning of the extent. This lets you separate positioning commands from editing commands in different device functions. Such granulation of a requested action makes each device function less specific and increase its reusability and efficiency.

The available merger commands are:

no operation	OVERLAY	(Table Text Center)REPLACE
* or #	PAINT	(Table Text Center)SCAN
FIND	POSITION	(Table Text Center)SKIP

INSERT	QUIT	(Table Text Center)UPDATE
MARK		(Table Text Center)

Command Syntax

A merger command has the following structure:

)	[<command verb>	[<*parameter*> = <*value*>]. . .]

where:

)	In column 1, identifies this as a merger command line rather than a source code line.
command verb	Identifies the main action. In general, each command has a three-letter abbreviation, which is shown preceded by | in the following command descriptions, for example: {INSERT | INS}.
parameter	Clarifies the main action.
value	Is a degree of the clarification and can be up to 60 characters.

The following syntax rules apply:

- One or more blanks are required as delimiters between the major structural parts.
- Each merger command must be coded on one line.
- Command text is not case sensitive unless otherwise noted.
- If a parameter value contains a blank, enclose it in either ' (single quotes) or " (double quotes).
- If a parameter value contains an apostrophe (single quote), either enclose it in " (double quotes) or enclose it in ' (single quotes) and duplicate the apostrophe.
- Any number of blanks are allowed between the parameter, equal sign, and value.
- Parameter names for all merger commands can be abbreviated using the first letter; for example, *COLUMN*= can be abbreviated as *C*=.
- Two special comment parameters, * and #, let you insert comments anywhere on a merger command. They are available on all merger commands and can appear multiple times on the same command. For example,

```
)  # Conditional change for DSPFIL processing (full-line comment)
) SCAN  * =  'Verify type'  FOR = TYPE:DSPFIL # = To_set_condition
) SKIP THROUGH = next  * = "Check condition"  IF = failed LAST= scan
    <device user source data lines comprising the patch>
) QUIT # = 'Exit, because the job done'
) MARK TAG = next
```

Alphabetical List of Merger Commands

or # (Full-line Comment)

(asterisk) or # (pound sign) indicate that the entire line is a comment.

(asterisk) or # (pound sign) indicate that the entire line is a comment.

)	{ * | # }	[< comment >]

Full-line comment commands are not counted by the SKIP command.

No Operation

The No Operation command consists of only a) in column one. It is counted when skipped using the SKIP command.

)

Use it for auxiliary purposes such as to indicate the end of a template group for the OVERLAY command.

FIND

FIND searches the generated source for the search string specified by the TEXT operand. The search starts from the current line and stops on the first occurrence of the found text or on the last line in the current device source extent. The search string is case sensitive.

)	{FIND | FND}	TEXT=<text>

INSERT

INSERT adds text after the current line. The added text starts immediately after the command and is interrupted by any line with) in column 1.

)	{INSERT | INS}

If the insert is successful, the current line is set to the last line added. If no lines were inserted, the current line remains unchanged.

MARK

MARK defines a label that can be referred to on the SKIP command to delimit a group of skipped commands. Place the MARK command after the last command to be skipped in the device user source. The label need not be unique; the choice is up to you.

)	{ MARK | MRK }	TAG = < tag name>

OVERLAY

OVERLAY uses the following user source lines as a template group to overlay the corresponding columns of the current line in the generated source. Each template line can potentially change columns 1 to 72. In case of a conflicting override by several templates, the result of the last one remains in effect.

By definition, characters in the generated source line that correspond to a blank character in the template line are not changed. To replace a character in the source line with a blank, assign a character to represent a blank on the template using the BLANK parameter. This character overlays the corresponding character in the source line with a blank.

)	{OVERLAY | OVR}	[BLANK = <character>]

Examples

The following lines substitute H for the character in column 10 of the current line in the generated source. All other characters are left unchanged.

```
) OVERLAY
                                        H
```

The following lines substitute a blank for the character in column 9 and an H for the character in column 10 of the current line in the generated source. All other characters are left unchanged.

```
) OVERLAY                               BLANK=%
                                        %H
```

PAINT

PAINT modifies the color of all generated lines for the given extent when the basic DSPF/PRTF source member is viewed. Painting the original device source extent clearly identifies the related generated source and is recommended before you create and attach device user source functions. Only lines produced by the CA 2E device generator are affected by this command.

)	{PAINT | PNT}	COLOR={CLEAR(CLR) | RED | GREEN(GRN) | WHITE(WHT) | TURQUOISE(TRQ) | YELLOW(YLW) | PINK(PNK) | BLUE(BLU) }

CLEAR removes any previous color. The abbreviation of a color in parentheses is equivalent to the main name.

Note: For all colors, the PAINT command paints only *unchanged* lines within the processed extent. For example, suppose an extent of source lines was painted yellow and then its second line was changed using the UPDATE command. If the entire extent is then painted blue later in the device user source, line two remains yellow whereas all others appear in blue.

POSITION

POSITION explicitly changes the current line to the requested location:

)	{POSITION | POS}	[TO={NEXT | FIRST | LAST}]

In case of a conflict with the requested value, the current line location remains unchanged; for example, specifying LINE=NEXT for the last line.

QUIT

QUIT unconditionally stops processing of the device user source. No parameters other than comments are available

)	{QUIT | QIT}

REPLACE

REPLACE substitutes one or more lines in the generated source with source statements in the device user source function. The deleted lines are not physically removed but are commented out and painted in red. They are excluded from the normal processing and are invisible to any subsequent FIND commands.

The inserted lines are the ones located between the REPLACE and the next command.

)	{REPLACE | RPL}	LINES={1 | <1–999>}

The actual number of lines after the current line may be less than the specified <number>. The added lines are always inserted AFTER the current line.

SCAN

SCAN searches the current line for the string specified by the FOR parameter.

)	{SCAN | SCN}		[FOR = <text>] [OCCURRENCE = {FIRST | LAST | <nn>}]
where:	<text> <nn>	Can be up to 60 characters long and is case sensitive. Is the occurrence (up to 70) of the FOR text being scanned for.	

SKIP

SKIP skips a specified group of subsequent commands in the device user source depending on whether the search run by the command identified by the LAST parameter failed or was successful. The skipped group begins with the command following the SKIP command and ends with the command identified by the COMMANDS or THROUGH parameters.

)	{SKIP | SKP}		[COMMANDS = {ALL | <nn>}] [THROUGH = <tag name>] [LAST = {FIND | SCAN }] [IF = {FAILED | SUCCESSFUL}]
where:	nn <tag name>	Is the number of commands to skip Is a label within the device user source defined by the MARK command to delimit the group of commands to be skipped.	

Notes

- Which commands to skip is determined by the COMMANDS and THROUGH parameters. These parameters are mutually exclusive:
  - The COMMANDS parameter skips either a specified number of merger commands (<nn>) or all remaining commands (ALL) in the device user source.
  - The THROUGH parameter specifies the label that delimits the end of the group of commands to be skipped. The label is defined by the MARK merger command, which must appear after the SKIP command. If the label is not unique, the first occurrence is used.
- When to skip a group of commands is determined by the LAST and IF parameters.
  - The LAST parameter specifies whether the SKIP command action depends on the result of the previous FIND (line search within entire generated extent) or SCAN (column search within the current line) command. FIND is the default.
  - The IF command specifies whether the SKIP command action depends on the success or failure of the command identified by the LAST parameter.

  Possible results are:

Value	FIND	SCAN
FAILED	SKIP the specified commands if the TEXT value was not found in the current extent in the generated source.	SKIP the specified commands if the FOR value was not found on the current line.
SUCCESSFUL	SKIP the specified commands if the TEXT value was found in the current extent in the generated source.	SKIP the specified commands if the FOR value was found on the current line.

- Full-line comments are not counted while skipping.
- Inserted data lines are considered as part of the preceding merger command and are not independently counted.

UPDATE

UPDATE replaces a portion of text in the current line. It first locates the text to be updated within the current line using the SUBSTRING and COLUMN parameters. It then replaces this text with the text specified by the BY parameter. If the length of the text to be updated differs from the length of the replacement text (the default length), use the LENGTH parameter to specify the number of characters to be replaced.

)	{UPDATE | UPD}		[SUBSTRING = <updated text>] [COLUMN = {1 | * | <nn> }] BY = <updating text> [LENGTH = [assign the value for mm in your book]]
where:	nn * mm	Is the column number where text to be updated starts. The default is 1. The column located by the previous SCAN. Is length of text to be replaced; defaults to length of text specified by the BY parameter.	

Notes

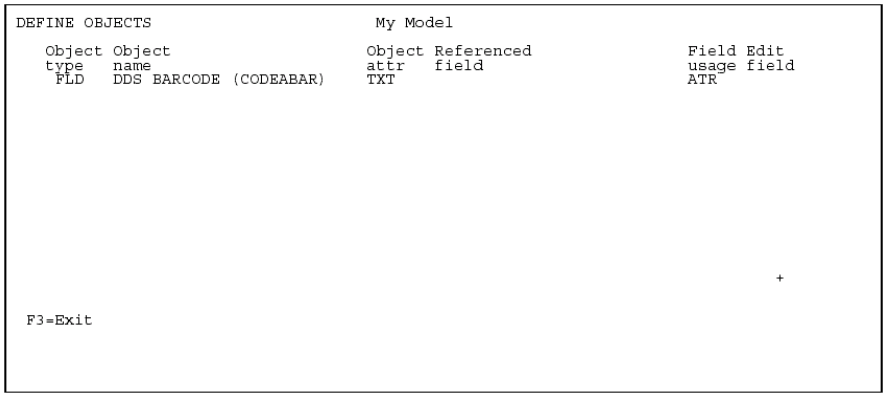
- If SUBSTRING is omitted, the COLUMN parameter indicates the beginning of the text to update.
- If SUBSTRING is specified, scanning starts from the position defined by the explicit or default value of the COLUMN parameter.
- If SUBSTRING is specified and the scan for it failed, the current line is not updated.
- To replace multiple occurrences of the specified text in the current line, first use SCAN to locate the beginning of the specified occurrence of the replaced text. Then use UPDATE with COLUMN=*, which sets the column position to the position referred to by the last SCAN.

Device User Source Example

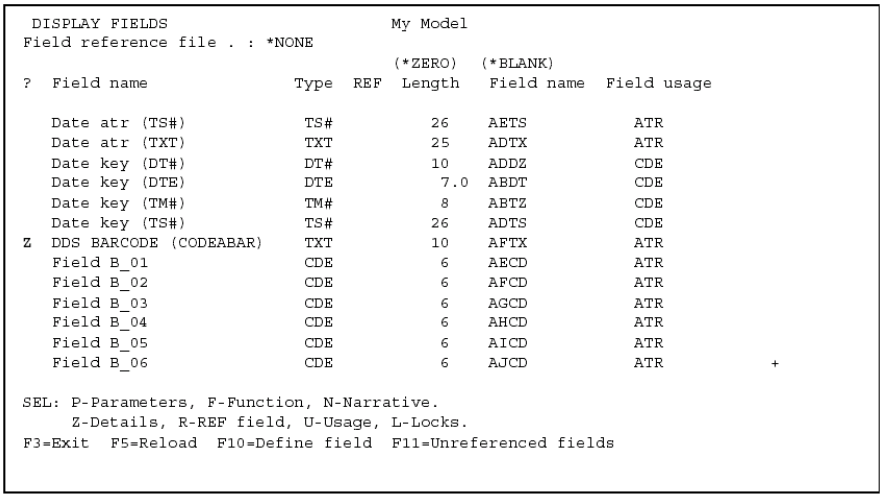
This example shows how to:

- Attach device user source to a field, which is then inherited by all device entries derived from this field.
- View and work with attached device user source from a device design
- View attached user source in the generated source of a device function; for example, a Print File (PRTFIL) function.

  Create a field; for example, DDS BARCODE (CODEABAR).



  The following steps show how to attach a device user source function to this new field. Type **Z** against the field.



Press Enter to display the Edit Field Details panel.

```
 EDIT FIELD DETAILS                        My Model
Field name . . . . . : DDS BARCODE (CODEABAR)    Document'n seq. . :
Type . . . . . . . . : TXT                                Field usage: ATR
Internal length. . :    10 Data type : A                  GEN name: AFTX
                           K'bd shift:    Lowercase : Y
Headings. . . . . . :-                                  Old DDS name:
  Text . . . . . . . . . . : DDS BARCODE (CODEABAR)
  Left hand side text. : DDS BARCODE (CODEABAR)
  Right hand side text : Text
  Column headings. . . : DDS BARCODE
                         (CODEABAR)

Control . . . . . . :-
  Default condition :     *NONE

  Check condition . :     *NONE
  Valid system name. . :    Mandatory fill . . . :


 F9=Conditions   F11=Field user source          F20=Narrative    F24=More keys
```

Press F11 to display the Field-Level Device Functions panel.

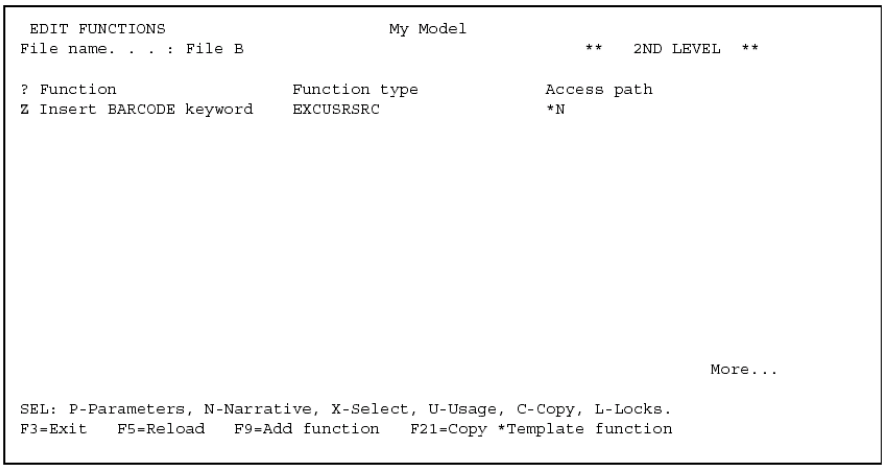
Field name

```
FIELD-LEVEL DEVICE FUNCTIONS          My Model

                                      Field  . . . : DDS BARCODE (CODEABAR)

  Device                                                          Applied
? Lang. [for] Device function     Associated with file            sequence









SEL: E-Edit device function source,  D-Detach function

F3=Exit        F5=Reload        F9=Attach function        F17=Services
```

Press F9 to define and attach the device user source function.

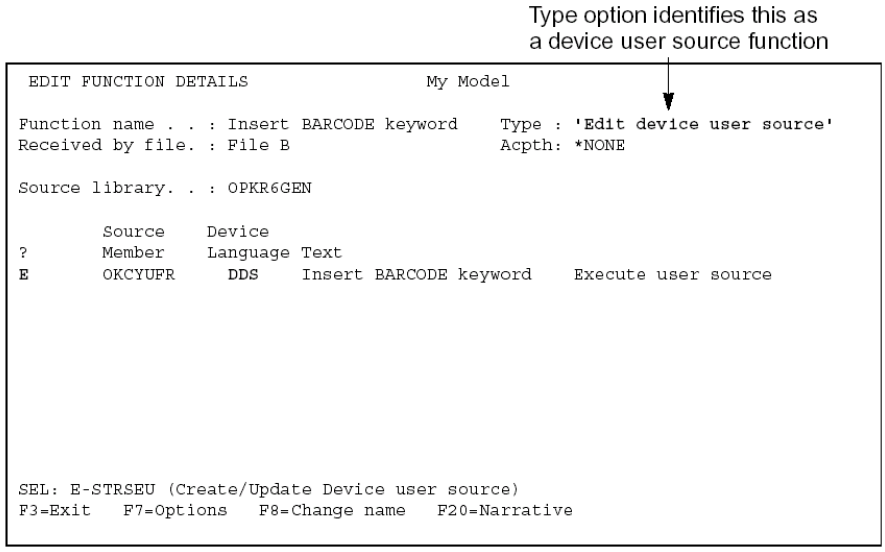
```
FIELD-LEVEL DEVICE FUNCTIONS          My Model

                                       Field  . . . : DDS BARCODE (CODEABAR)

  Device                                                          Applied
? Lang. [for] Device function     Associated with file            sequence

              ..............................................
              :  ATTACH DEVICE FUNCTION                     :
              :                                             :
              : Function file : ?                           :
              : Function. . . : ?                           :
              :                                             :
              : F3=Exit                                     :
              :                                             :
              :                                             :
              :.............................................:


SEL: E-Edit device function source,  D-Detach function

F3=Exit        F5=Reload        F9=Attach function        F17=Services
```

Specify the file to which the new function is attached, in this case File B, and create a new EXCUSRSRC function. Type **Z** against the new function.

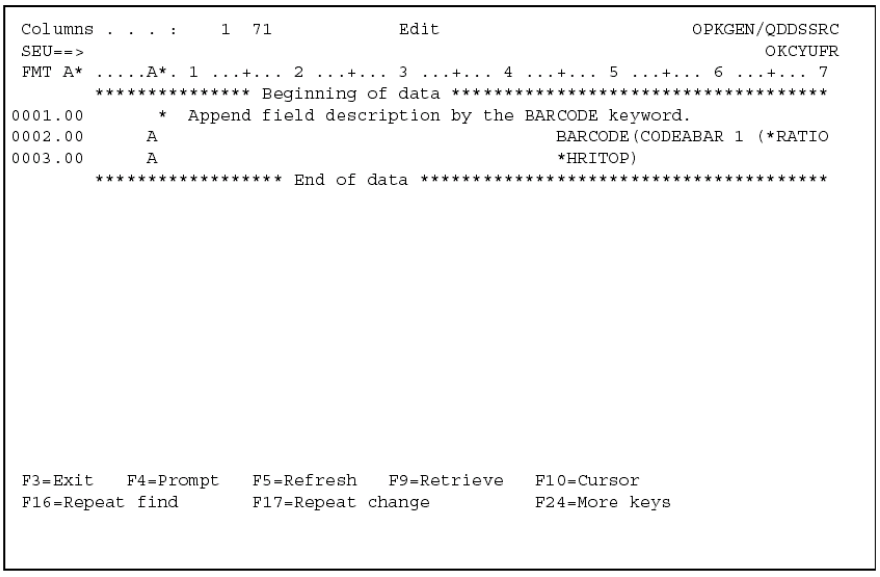


Press Enter.

Type **DDS** instead of the current HLL name and press Enter. Note that the Type option changes from ‘Execute user source’ to ‘Edit device user source.’ Type **E** against the function to invoke SEU for the source member.



Type the DDS source statements that you want applied to device entries derived from the DDS BARCODE (CODEABAR) field.



Exit and save the function. On the Edit Functions panel, select the device user source function.

```
 EDIT FUNCTIONS                        My Model
File name. . . : File B                                    **   2ND LEVEL  **

? Function                  Function type             Access path
  Device 7                  Execute user source       *NONE
  Device 8                  Execute user source       *NONE
  Device 9                  Execute user source       *NONE
  Edit File B               Edit file                 Retrieval index
X Insert BARCODE keyword    Execute user source       *NONE
  PRTOBJ File B             Print object              Retrieval index
  PRTOBJ:Report lvl(Yellow) Execute user source       *NONE
  SDF example for File B    Execute user source       RSQ for file B
  Select File B             Select record             Retrieval index




                                                                        More...

SEL: P-Parameters, N-Narrative, X-Select, U-Usage, C-Copy, L-Locks.
F3=Exit    F5=Reload    F9=Add function    F21=Copy *Template function
```

Press Enter.

```
FIELD-LEVEL DEVICE FUNCTIONS          My Model

                                      Field  . . . : DDS BARCODE (CODEABAR)

  Device                                                              Applied
? Lang. [for] Device function     Associated with file              sequence
    DDS Insert BARCODE keyword    File B                              10









                                                                      Bottom
SEL: E-Edit device function source,  D-Detach function

F3=Exit        F5=Reload        F9=Attach function        F17=Services
'Insert BARCODE keyword' has been attached.
```

From this panel you can

- Use the E option to edit the device user source
- Use the D option to detach a device user function

1. Press F3 to exit to the Field-Level Device Functions panel. The device user source function, Insert BARCODE keyword, is now implicitly attached to derived entries for the DDS BARCODE (CODEABAR) field on all device designs and is applied the next time the device designs are regenerated.
2. Press F3 to exit the Edit Field Details panel.

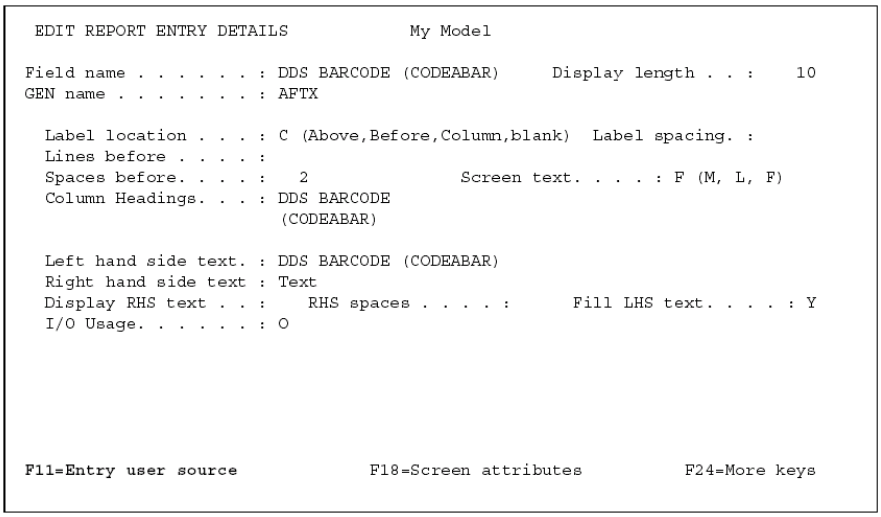
Enter the device design of an existing PRTFIL function.

```
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO Print Product




Product Product Name              Product
Code                              Type

OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO
OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO
OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO

Final totals

** END OF REPORT **
```

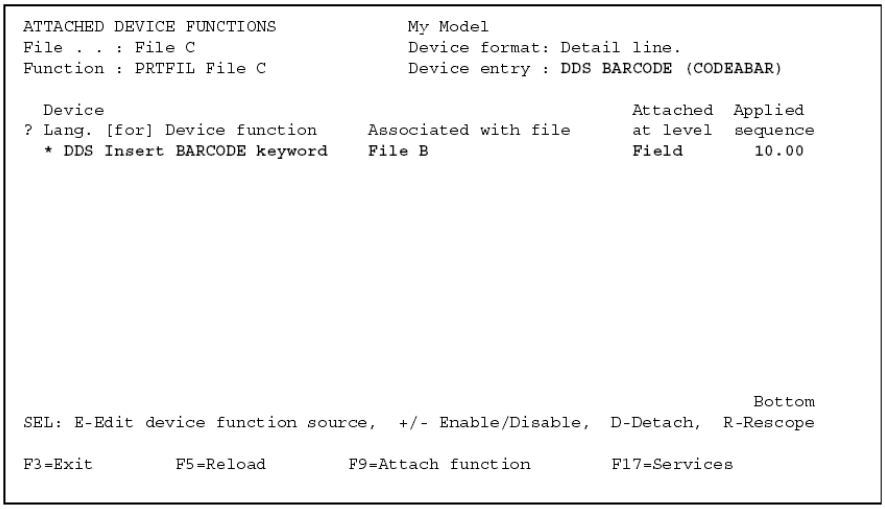
As an example, you can add DDS BARCODE (CODEABAR) as a function field to the device design (press F19).

```
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO Print Product




Product Product Name              Product   DDS BARCODE
Code                              Type      (CODEABAR)

OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO  OOOOOOOOOO
OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO  OOOOOOOOOO
OOOOOO  OOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOO  OOOOOOOOOO

Final totals

** END OF REPORT **
```

Position the cursor on the DDS BARCODE (CODEABAR) entry and press Enter to display the Edit Report Entry Details panel.

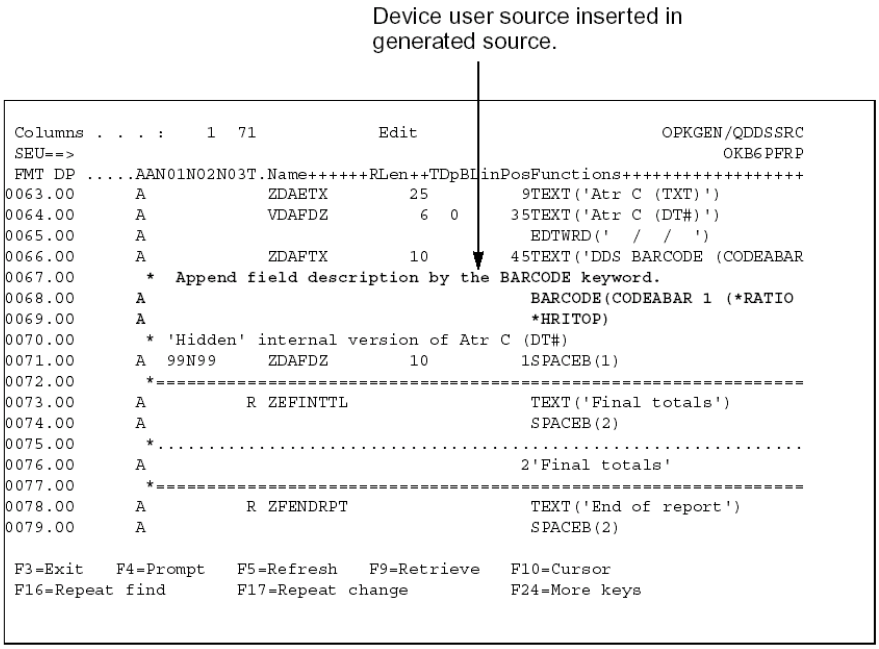


Press F11 to view the device user source attachments for the entry.



3. Press F3 to exit.

   View the generated result.



Copying Functions That Contain Attached Device User Source

- Device-level device user source is always copied even if the target file (ACP) is not the same as the source file.
- Screen, report, and format level device user source is copied unless the corresponding entity in the target function is excluded due to a change of function type.
- Entry-level device user source is copied unless the corresponding entry in the target function is excluded due to an access path change.

Reference Field

A reference field inherits all device user source attached to the original field when it is created. Subsequent changes to device user source attached to the original field are not reflected in the reference field.

Documenting Functions

The listing produced by the Document Model Functions (YDOCMDLFUN) command indicates whether functions contain device user source. Specifying either *BASIC or *FULL for the PRTDEVDTL parameter provides a separate summary of attached device user source functions.

Guidelines for Using Device User Source

The following guidelines will help you decide which attachment level to specify when attaching device user source.

Attachment Levels Are Not Hierarchical

Before attaching Device User Source, it is important for you to understand what each of the four device user source attachment levels cover.

- Device (device-level attachment)
- Screen/Report (screen- or report-level attachment)
- Format (format-level attachment)
- Entry (entry-level attachment)

The attachment levels do not form a hierarchy. For example:

- Attaching a section of device user source at format-level does not enable it over a field within that format.
- Attaching device user source at device-level only has relevance at device-level, not at any *lower* level.

Note: This discussion does not apply to the field attachment level, which provides a generic way to apply user source to all derived entries for a field.

Understanding Extents

When using device user source, it is helpful to think of the CA 2E generated source as a collection of independent source text extents rather than as a single source member. When device user source is merged with the CA 2E source, it is merged with an extent, not with the source member as a whole. You select the appropriate extent when you specify the attachment level.

The following points are important to keep in mind regarding extent:

- Each extent contains a different section of code that corresponds to one of the attachment levels.
- Each extent is entirely separate from every other one.
- One attachment level can be resolved into several non-contiguous extents. For example, a multi-line column header can result in up to four extents.

Note: The following examples for DDS are true for SDF as well.

After all device user source is merged, the result is a DDS source file suitable for input to the DDS compiler.

Example:

A typical PMTRCD display file might consist of the following extents, in top-down order:

Extent	Description
Device extent	
Format extent	(key screen record format)
Entry extent	(key screen 1st field entry)
Entry extent	(key screen 2nd field entry)
Format extent	(detail screen record format)
Entry extent	(detail screen 1st field entry)
Entry extent	(detail screen 2nd field entry)
Entry extent	(detail screen 3rd field entry)
Entry extent	(detail screen 4th field entry)
Entry extent	(detail screen 5th field entry)
Screen extent	(detail screen 5th field entry)

A common error is to attach device user source for a DDS field (entry-level extent) to the DDS record (format-level extent) that contains the selected field. If the device user source contains a FIND merger command to locate the selected field, the field is not found since the scope of a FIND is restricted to the specified extent and the format extent contains only record data definition lines.

By definition, FIND sequentially checks all available lines after the current one and stops at the last one making it the new current line. The search has actually failed, but if SKIP is not used, the following INSERT command places the patch after the current line, for example, at the end of DDS format definition and not in the DDS field definition as intended.

Visualizing Extents

If it is not clear which attachment levels to use for a particular change, you can make a self-educating demonstration for each external function type. To do so, produce:

- A set of device user source functions, one for each attachment level. Each function contains only a PAINT command and comments defining the boundaries of the extent.
- A set of simple samples of each external function type, containing no more than one or two fields per format.

You then attach the device user source functions to your sample external functions to cause the device source comprising each extent to display in a separate color. The color indicates the attachment level to which the extent belongs.

In the future when you are attempting to attach device user source to one of your complex functions you can use these sample, painted functions as a reference to help you locate the correct attachment point for your patch within the device design of the function.

The following steps are basic *recommendations* for painting generated device source.

1. Define DDS and SDF files of STR type. For better maintainability, keep all *painting* functions (see examples below) built over the two files accordingly.
2. Create ‘painting’ device user source functions, one per attachment level (Device, Screen/Report, Format, and Entry) and per device language (DDS, SDF).
3. Name the ‘painting’ functions to reflect the attachment level that is to be painted.
4. Type in similar user source that inserts the boundary comments (<<<...>>>) and paints the entire extent in a selected color. Use unique colors for each device user source function within the scope of device language.
5. Attach ‘painting’ device user source functions to *all* available attachment points of the selected external function. In order to cover Header/Footer entities, create a special version of the Header/Footer function in advance, and paint it as you would paint any external function.
6. Generate the selected external functions and enjoy seeing the fully painted device source produced by CA 2E. This should assist you in understanding what extents are and how to use the power of the device user source feature.

Examples of 'Painting' Functions

Device User source function 'Paint FIELD source' built over DDS file:

```
    * <<<<<<<<< The beginning of FIELD extent >>>>>>>>>>
) POSITION TO=LAST
    * <<<<<<<<< The end of FIELD extent >>>>>>>>>>>>>>>>
) PAINT COLOR=BLUE
```

Device User source function 'Paint FORMAT source' built over SDF file:

```
# <<<<<<<<< The beginning of FORMAT extent >>>>>>>>>>
) POSITION TO=LAST
# <<<<<<<<< The end of FORMAT extent >>>>>>>>>>>>>>>>
) PAINT COLOR=RED
```

Contents of Extents

Device Extent

The Device extent covers the T*, Z*, and H* lines that are automatically generated for CA 2E source, plus any file-level lines that are automatically generated, such as the definition of the print key, help key, and so on. In other words, it covers every source line from the top of the source member to the first record format definition line. For example,

```
T* Test EDTRCD2 for DDS SRC Edit record(2screens)
Z* CRTDSPF
Z* RSTDSP(*YES)
H* MEMBER-ID: UUAXE2R#
*
H* Generated by :SYNON/2 Version: 1037
H* Function type:Edit record(2 screens)
*
H* Company      :RMHR6MDL
H* System      :RMHR6MDL
H* User name    :RMH
H* Date       :01/23/98 Time :15:42:00
H* Copyright    :RMHR6MDL
*=============================================================
M* Maintenance  :
*=============================================================
A                    INDARA
A                    PRINT(YPRTKEY$)
A                    ALTHELP(CA01)
A                    ALTPAGEXCUSRPGM(CF07)
A                    ALTPAGEDWN(CF08)
A                    CHGINPDFT
A                    HELP
A                    HLPTITLE('Test EDTRCD2 for DDS SRC-
A                    - Help')
A                    HLPPNLGRP('UUAXE2RH' UUAXE2RH)
A*Window borders definition
A                    WDWBORDER((*COLOR BLU)
A                    )
```

Note: Device user source containing format-level data, like non-standard command key usage, such as CA01(03 'Exit'), should normally be attached to a record format. However, if the device user source applies to all formats within the file it can be attached at device-level.

Format Extent

The Format extent covers only those sections of the DDS source file that refer to format-level code, such as Command key definition, SETOFs, Help specifications, and cursor location specifications. It does not cover any of the fields that exist within that format. For example,

```
A     R #RCDKEY             TEXT('KEY SCREEN')
A                           SETOF(98 'Force input format')
A                                  BLINK OVERLAY
A                                  INDTXT(86 'Enable PUTOVR')
A 86                        PUTOVR
A 04
AO 25                       CSRLOC(ZZCSRW XXCSCL)
A                                  INDTXT989 ''ADD' mode')
A* Command keys
A                                  VLDCMDKEY(29 'Valid command key')
A                                  CA03(03 'Exit.')
A                                  CF09(09 'Change mode.')
A                                  CF05(05 'Reset.')
A                                  CF04(04 'Prompt.')
A* SETOFFS.............................
A                                  SETOF(99 'Global error flag')
A                                  SETOF(31 'Invalid: #1ADCD')
* ..............................
* Help specifications
A      H                    HLPARA(*NONE)
A                                  HLPPNLGRP('INTRO' UUAXE2RH)
A      H                    HLPARA(*NONE)
A                                  HLPPNLGRP('KEY' UUAXE2RH)
A      H                    HLPARA(*NONE)
A                                  HLPPNLGRP('FUNCK" UUAXE2RH)
A      H                    HLPARA(*NONE)
A                                  HLPPNLGRP('DATAK' UUAXE2RH)
A                                  HLPEXCLD
* ..............................
* Reposition cursor to where?
A       ZZCSRW     3 0H     TEXT('Cursor row')
A       ZZCSCL     3 0H     TEXT('Cursor Column')
```

Entry Extent

The Entry extent covers only those sections of the DDS source file that describe the characteristics of specific device entries (fields) within a format. For example,

```
A       #1ADCD     3 B 415 TEXT('Key code')
A                            CHECK(FE)
A N25                        OVRDTA
A 31                         DSPATR(R1)
A N31                        DSPATR(UL)
A 31
AON31N98N99                  DSPATR(PC)
A N25                        OVRATR
```

Screen Extent

The Screen extent covers the message subfile and confirm prompts that appear at the base of each screen. For example,

```
.*=================================================================
.     R #CONFIRM             TEXT('Prompt confirm')
.                                    VLDCMDKEY(29)
.                                    OVERLAY PROTECT PUTOVR CLRL(*NO)
.               24 64        'CONFIRM.'
.      ##CFCD    1  H        TEXT('*CONFIRM')
.      V#CFCD    1  B 24 73       TEXT('*CONFIRM : External Image')
.                                    CHECK(ER)
.                                    DSPATR(HI UL)
. 96                         ERRMSGID(Y2U0014 Y2USRMSG)
.               24 75        '(Y/N)'
.*=================================================================
.     R #MSGRCD              TEXT('Program messages')
.                                    SFLCTL(#MSGRCD)
.       MSGKEY               SFLMSGKEY
.       ##PGM                SFLPGMQ
.*=================================================================
.     R #MSGCTL              TEXT('Program messages')
.                                    SFLCTL(#MSGRCD)
.                                    SFLPAG(01) SFLSIZ(03)
. 86                         OVERLAY PUTOVR
.                                  SFLINZ SFLDSP SFLDSPCTL
. 25
.ON25                        SFLEND
.      ##PGM                 SFLPGMQ
.*=================================================================
```

Device Source Extent Stamp (DSES)

The number and content of generated extents for a particular device entity varies from one function type to another. Prior to Release 6.1, separate device user source functions were required to produce a similar change to functions of different types.

To resolve this problem and expand the power of device user source, the generators insert an identifying comment before each extent that is part of a device entity that has device user source attached. The comment contains the following information: attachment level, header/footer type, program name, and function, entity, screen, format, and entity types. This comment is called the Device Source Extent Stamp (DSES) and consists of the following components.

Note: Not all extents have all components.

Component	Description	Valid Values
A@L:n	Attachment level; read as Attached at Level:	E (entry), F (format), S (screen), R (report), and D (device)
HDR:nnnnn	Header/Footer type	POPUP, F/SCR, PRINT
FUN:nnnnnnnnnn	Program name	For example, UUABEFR
TYPE:nnnnnnn	Function type	EDTRCD, DSPTRN, PRTFIL, etc.
SCR#:n RPT# :n	Screen/Report number	1, 2, 3,4
FMT:nnn	Format type	RCD, CTL, HDR, FTR, etc.
ENT:n	Entity type	F (field description) 1, 2, 3, L, R (column headers) Special field values: P (program name), ! (screen title, $ (selection text), # (command key text)

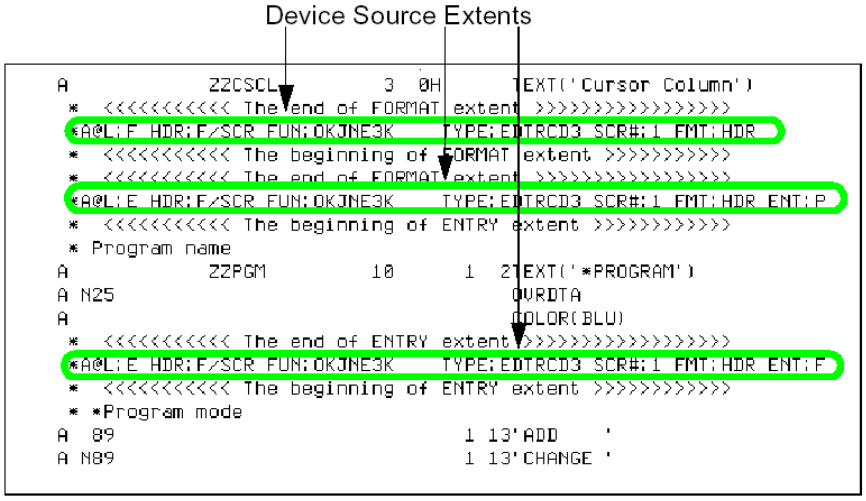
By specifying values contained in the DSES on a SCAN command, you can conditionally test whether or not the processed extent should be modified. For example, to attach user source to a field description rather than to a field column header, scan for ‘ENT:F’ rather than ‘ENT:1’ or ‘ENT:L’.

Note: DSES is always the first line in any extent. To ensure that the current line is positioned at the DSES, include a POSITION TO=FIRST command in the user source.

The DSES is not protected from change, so be careful not to modify it accidentally.

Examples of Device Source Extent Stamp

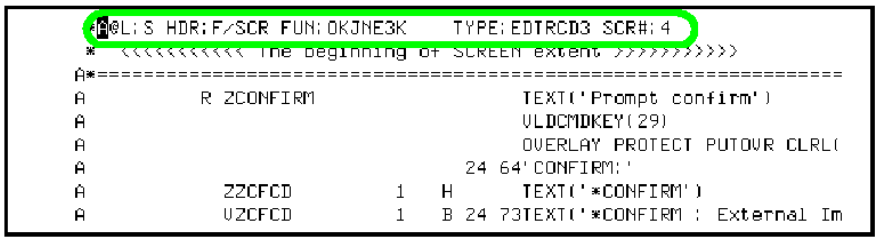
This shows DSES for several contiguous extents along with comment boundaries of each extent (<<<...>>>) that were embedded by the attached ‘painting’ functions.



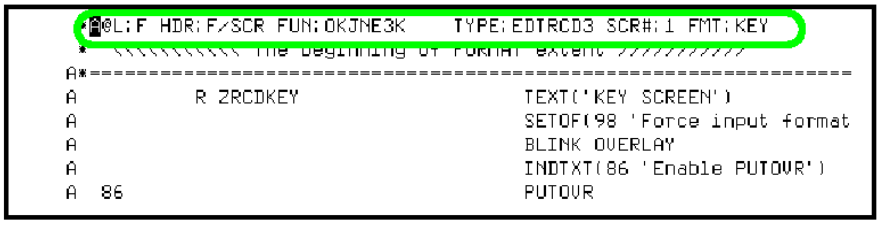
This shows the device source header including pre-processor directives *Y, *Z, and so on. The stamp does not have screen-, format- and entry- components.

```
*A@L:D HDR:F/SCR FUN:OKJNE3K TYPE:EDTRCD3
*  <<<<<<<<<<< The beginning of DEVICE extent >>>>>>>>>>>
T* EdtRcd3 test for DSES     Edit record(3 screens)
Z* CRTDSPF
Z* RSTDSP(*YES)
H* MEMBER-ID: OKJNE3KD
```

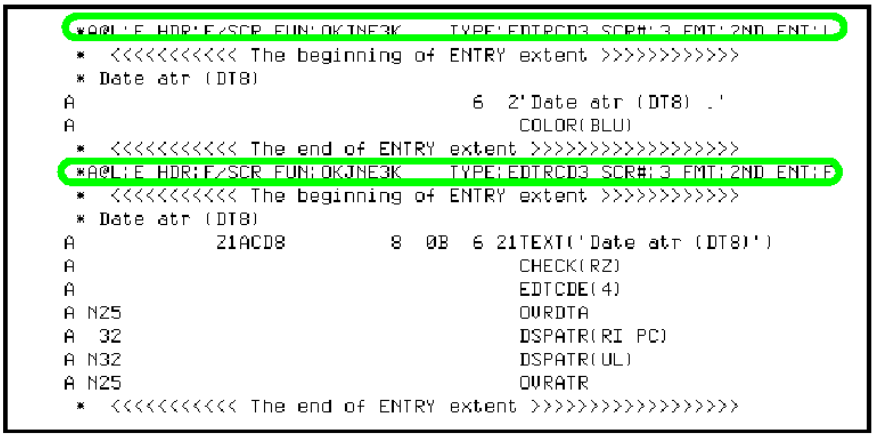
This shows the Detail Screen 3 (SCR#:4) containing the Confirm prompt window description.



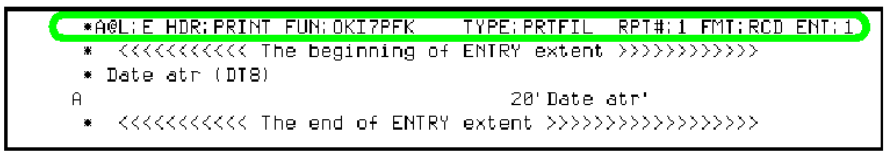
This shows the Prompt Key Details format (FMT:KEY) on the Prompt Key screen (SCR#:1).



This shows the Date atr (DT8) field of the Detail Format 2 format (FMT:2ND) on the Detail Screen 2 screen (SCR#:3) presented by two extents describing Left Hand Side Text (ENT:L) and the field itself (ENT:F), respectively.



This shows the same field belonging now to the Detail line format (FMT:RCD) on the only report (RPT#:1) of the function (TYPE:PRTFIL) with selected Standard Report Header/Footer (HDR:PRINT).



Chapter 10: Modifying Action Diagrams

This chapter describes the components that make up an action diagram, how to use the action diagram editor, and how to edit a function's action diagram.

This section contains the following topics:

Understanding Action Diagrams

Action diagrams record the basic constructs that make up a procedure. The action diagram is used to specify the procedural steps that make up a CA 2E function. These procedural steps encompass a list of actions; each action can either be a call to another function or a number of low-level built-in functions.

Depending on where you are in CA 2E, use one of the following sets of instructions to get to the action diagram of a function. These instructions are only provided here, in the beginning of this chapter. Other instructions in this chapter assume that you are already at the Edit Action Diagram panel.

When a parameter is being passed as an array there is a single subfile line that indicates an array being passed.

Note: If the called function's parameter interface is modified to toggle the parameter Passed as Array field from Y to blank, the behavior of the EDIT ACTION – FUNCTION DETAILS changes accordingly to match.

The Edit Database Relations Panel

To use the Edit Database Relations panel

1. Go to the function. At the Edit Database Relations panel, type **F** next to any relation for the file.

   The Edit Functions panel appears.

2. Go to the action diagram. Type **FF** next to the selected function.

   The Edit Action Diagram panel appears.

The Open Functions Panel

To use the Open Functions Panel

1. Go to the action diagram.
2. Type **F** next to the selected function at the Open Functions panel, and then press Enter.

   The Edit Action Diagram panel appears.

The Edit Function Details Panel

To use the Edit Functions Detail panel

- If you are at the Edit Function Details panel, press F5 to display the Edit Action Diagram panel.

The Display All Functions Panel

To use the Display All Functions panel

1. Go to Display Services.
2. From within CA 2E, press F17.

   The Display Services Menu appears.
3. Go to the list of all functions, select the Display all functions option, and then press Enter.

   The Display All Functions panel appears.
4. Go to the action diagram.
5. Type **F** next to the selected function and then press Enter.

   The Edit Action Diagram panel appears.

Specifying an Action in an Action Diagram

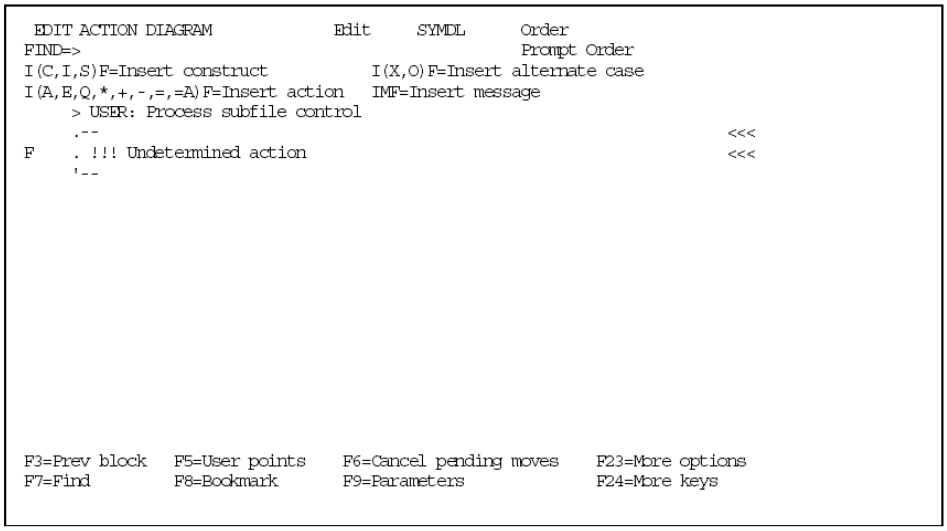
To specify an action use the Action Diagram Editor with this two-step process

1. Specify where in the action diagram you want the action to execute.
2. Specify the function details for the new action.

Adding an Action

To add an action

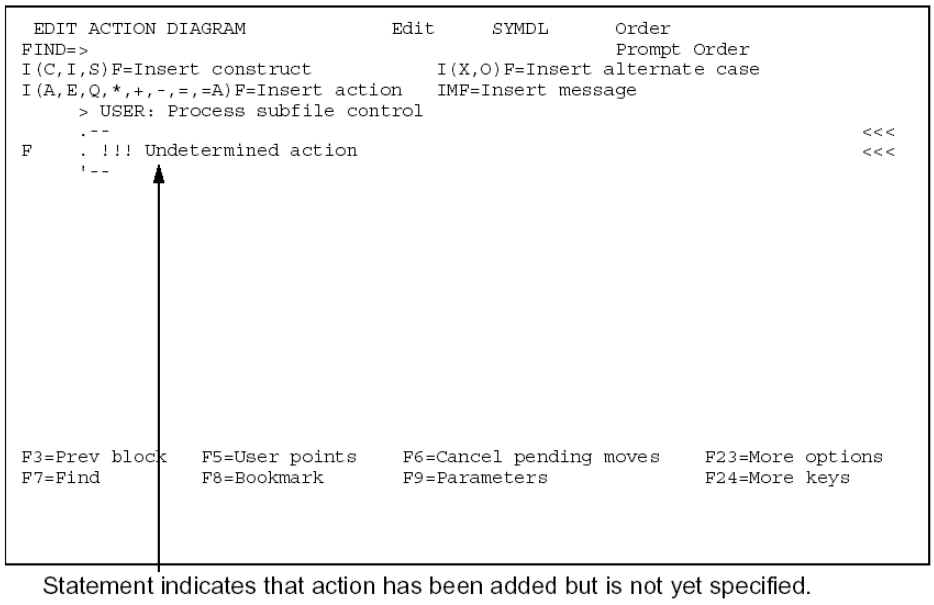
- Specify IA against the line in the action diagram where you want to add the action:



Specifying a Function as an Action

To specify the details for the new action

- Enter **F** against the line just added:



Alternatively, you can add or specify a function as an action in one operation by entering **IAF** in the subfile selection column.

Naming a Function as an Action

This provides a subsidiary display on which you can specify the name of the function that constitutes the action. The display is preloaded with a ? in the Function file and Function fields to facilitate inquiries for these fields.

For the File, you may enter one of the following special values. If the Function is specified as ?, you will be prompted to specify a function name based on the File.

*****

Use the specified function based on the current file.

?*

Prompts for all system files.

Blank

Uses the specified built-in function.

***A**

Uses the specified function based on the *Arrays file.

***F**

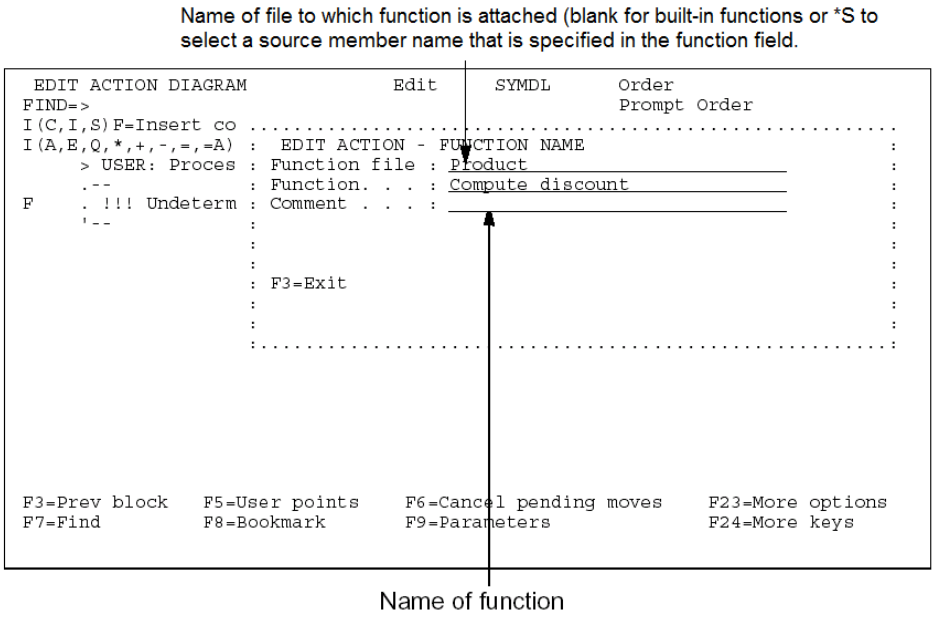
Uses the specified function field.

***M**

Uses the specified function based on the *Messages file.

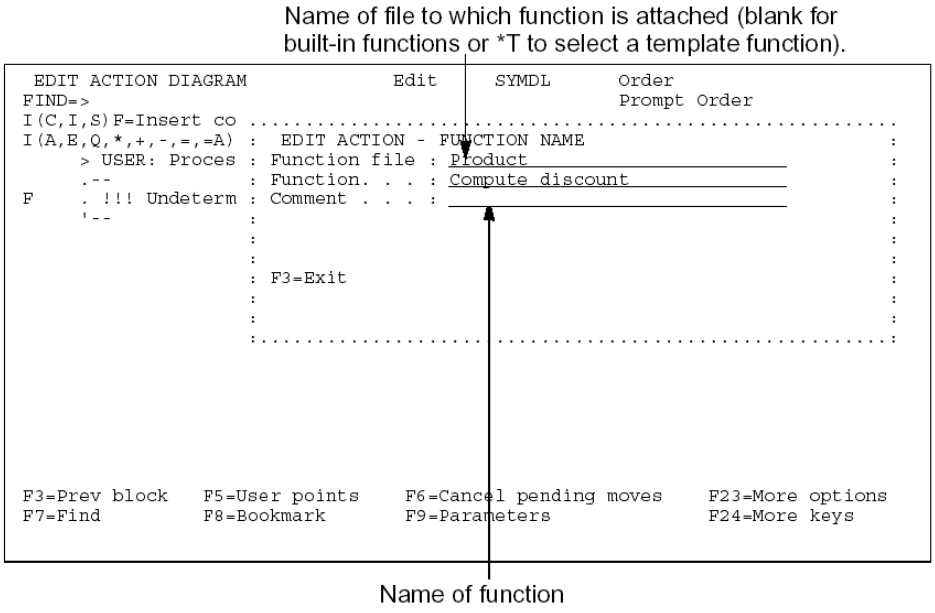
***I**

Uses the function whose source member name is specified in the Function field.



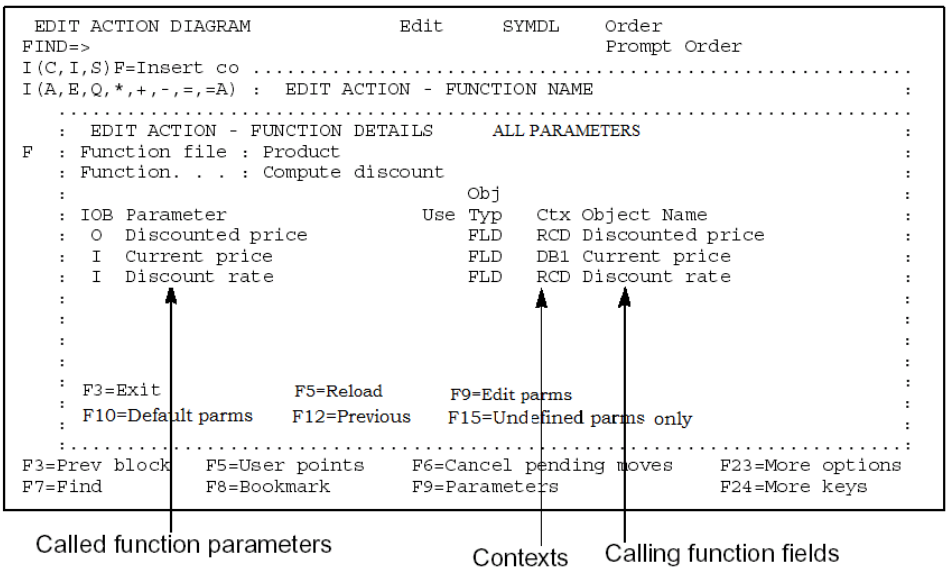
***T**

Uses the specified function based on the *Template file.



Specifying Parameters for an Action Function

If there are any parameters for the called function, you can specify which fields from which contexts within the calling function are to be passed as parameters. To do so, press F9.



If the function has more than eight parameters, a parameter selection field appears in the header. Press F15 to display undefined parameters. Press F15 again to display only those parameters that are defined with a context other than NLL. Press F15 again to display all parameters.

Note: Changes do not occur when pressing F15. The ENTER key must be pressed to confirm any changes to an action.

When Action Function parameters are undefined, the default model context appears in the upper right corner where you can change it. For example, if the model default is the WRK context you can change the value to LCL or NLL context and press F10=Default parms.

Calling a Function with a Parameter Passed as Array

When there is only one item passed as array, as in the above example, Ctx defaults to PAR.

For more information on parameter usage see the Parameter Usage matrix (see page 282).

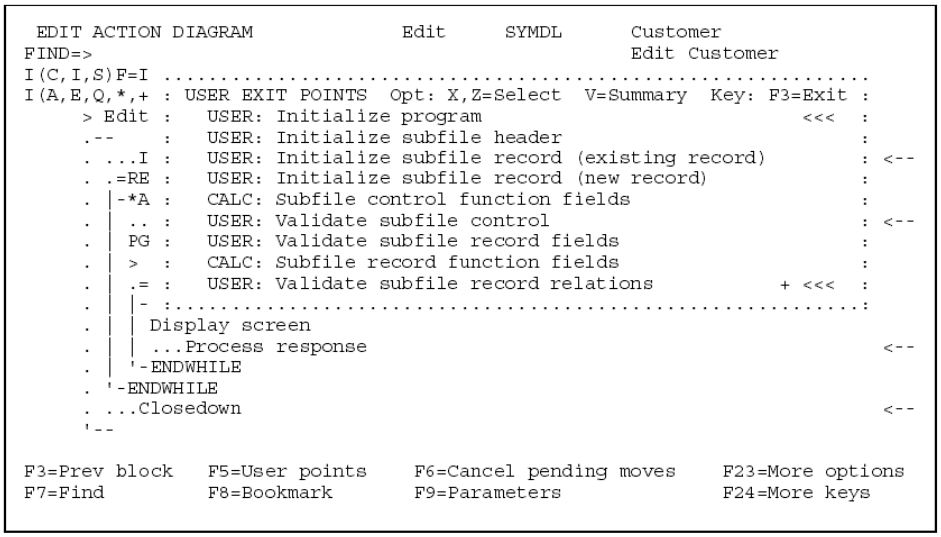
User Points

The majority of the CA 2E standard functions have default action diagrams. The exceptions are Execute User Source (EXCUSRSRC) and Execute User Program (EXCUSRPGM). Portions of the action diagram are essential to the program's function, and as such, are not alterable. However, you can insert logic into the action diagram to add processing that is specific to that function.

The areas that you can modify in the action diagram are called user points. The user points vary for each standard function and are accessed according to the function. User points are identified in the action diagram of a function by arrows, made up of a chevron and two dashes, in the right margin of the Edit Action Diagram panel.

To list and access the user points in an action diagram, press F5 to display the available user points for the function.

The Action Diagram User Point window appears.



If the user points contain user-defined action diagram statements, they are identified by three chevrons in the right margin of the window.

For more information on individual user points for each function type, see the Understanding Action Diagrams User Points topic, later in this chapter.

Understanding Constructs

Constructs are the basic building blocks of an action diagram. By combining different types of constructs, you define the procedural logic of an action diagram.

The action diagram allows those basic constructs, action and condition, to be combined into other types of constructs. The combination constructs are as follows:

- Sequential
- Conditional
- Iterative

CA 2E executes all actions in a bracket construct in order, from top to bottom.

The following is an example of the presentation convention for action diagrams.

```
> Process subfile                          <==TITLE
.--                                        <==SEQUENCE
: Read next changed SFL record
: .=REPEAT WHILE                           <==ITERATION
: |-Changed record found on SFl            <==CONDITION
  |
: | .-CASE
  |
: | |-RCD. *SFLSEL *Zoom                   <==CONDITION
: | |>USER DEFINED SELECTION                          <<<
: | |.--                                              <<<
: | |:.Display user details                <==ACTION  <<<
: | |:PAR: Date |: 'Date of Birth'
: | |'--                                              <<<
: | |-*OTHERWISE                            <==CONDITION
: | |>USER DEFINED LINE VALIDATION                    <<<
: | |.--                                              <<<
: | |:.-CASE                                          <<<
: | |:|-RCD.Date of birth * GT JOB.Job date           <<<
: | |:|.Send error message 'Invalid DOB'   <==ACTION
: | |:|PAR: Date|:'Date of Birth'
: | |:'-ENDCASE                                       <<<
: | |'--
: | '-ENDCASE
: |   Update SFL record
:     Read next changed SFL record

: '-ENDWHILE
'--
```

Sequential

Sequential statements are the simplest of the constructs. They specify a list of actions or other constructs that are executed in the order in which they are written. The logic that you insert in a sequential statement is normally executed as a subroutine within the HLL program.

Sequential statements are denoted by brackets that enclose dotted lines.

```
.—

: ...
: ...
: ...
‘--
```

Conditional

Conditional constructs allow actions to be conditionally executed within the action diagram logic. Conditional constructs generally conform to nested IF THEN ELSE logic statements or SELECT sets. Conditional constructs are called CASE blocks because they are denoted by CASE and ENDCASE statements. Conditions that are more complex can be constructed using compound conditions. You use CASE blocks to execute actions based on the condition being tested. Actions within a condition construct are generated as inline code.

Condition constructs are denoted by brackets that enclose broken vertical bars.

```
.-CASE

|-Order status is Held
| ...
| ...
| ...
‘-ENDCASE
```

Iterative

Iterative constructs represent repetitive logic that executes when a specific condition is true. The iterative statement is denoted by REPEAT WHILE and ENDWHILE statements. CA 2E implements the iterative construct as an HLL subroutine. You must define a controlling condition within the iterative loop to determine whether the logic is to be repeated.

Note: The actions within the iterative construct are executed only while the initial condition is true. This may require a preceding action to set the initial condition.

Iterative constructs are denoted by brackets that enclose solid vertical lines.

```
.=REPEAT WHILE

-Order status is Held
...
...
...
‘-ENDWHILE
```

Capabilities of Constructs

You can nest constructs. For example, you can insert a conditional CASE construct within a REPEAT WHILE construct; in this manner you test for a conditional value while the controlling condition is executed. You can nest sequential actions within any other construct.

```
.=REPEAT WHILE
| |-END OF CURRENT CUSTOMERS NOT REACHED
| |...Read customers
| |...
| |.-CASE
| |-Order value is *LT 100
| | ...
| | ...
| '-ENDCASE
| ...
'-ENDWHILE
```

You can exit a construct at any point within the processing logic by means of either one of two built-in functions: *QUIT or *EXIT PROGRAM.

- The *EXIT PROGRAM built-in function allows you to leave the current HLL program or CA 2E external function.
- The *QUIT built-in function allows you to leave the current subroutine logic of the sequential construct block or CA 2E internal function.

There are consequences for using *QUIT and *EXIT PROGRAM within constructs. Other constructs are implemented as inline code within the current subroutine. The *QUIT built-in function allows you to leave the current subroutine logic of the construct block. The *EXIT PROGRAM built-in function allows you to leave the current HLL program.

```
.—SEQUENCE

:  .Order status is Held
:  ...Review order
:  .-CASE
:  |...Order value is RELEASE
:  |  <--*QUIT
:  '-ENDCASE
:  ...
'--
```

For more information on user points, see Understanding Action Diagram User Points at the end of this chapter.

Understanding Built-In Functions

CA 2E built-in functions specify low-level operations that you can use within the user points in action diagrams to implement a specific field manipulation or to control an action within the action diagram.

Follow these steps to insert a built-in function at an action diagram user point.

1. Enter **IAF** next to the location in the action diagram where you want to insert the function. The Edit Action - Function Name window appears.
   - Leave the Function File option blank to default to the *Built in functions file.
   - Enter **?** for the Function option and press Enter to display a list of the built-in functions. Alternatively, you can enter the name of the built-in function.
2. Select the built-in function you want. The Edit Action - Function Name window appears. Press Enter to display the Edit Action - Function Details window and enter parameters for the built-in function you selected.
3. Press Enter to continue editing the Action Diagram.

Each of the built-in functions are listed alphabetically and described on the following pages.

Add

The *ADD built-in function specifies an arithmetic addition on two operands.

There are three parameters for this function type:

- Two input parameters, which are the two operands.
- One output parameter, which is the *Result field containing the result of the addition

All three parameters must be a numeric field type such as PRICE or QUANTITY.

CA 2E implements the *ADD built-in function as an ADD statement for all generators.

Example

This is an example of the *ADD built-in function

```
> USER: Process detail record
 .--
 :  WRK.Cost = RCD.Cost + RCD.Tax       <<<
 '--
```

Commit

The *COMMIT built-in function enables you to add your own commit points to a program that is executing under i OS commitment control. Commitment control is a method of grouping database file operations that allow the processing of a database change to be either fully processed (COMMIT) or fully removed (ROLLBACK).

There are no parameters for this built-in function.

CA 2E implements the *COMMIT built-in function as an RPG COMMIT statement, and as a COBOL COMMIT statement.

Example

The following is an example of the *COMMIT built-in function.

```
>  USER: Create DBF record

.--
:  .Call EDTRCD function                          <<<
:  .-CASE:                                        <<<
:  | -PGM.*Return code is Not blank                <<<
:  |  Rollback                                     <<<
:  |  <--QUIT                                      <<<
:  '-ENDCASE                                      <<<
:  Call EDTTRN function                           <<<
:  .-CASE:                                        <<<
:  | -PGM.*Return code is Not blank                <<<
:  |  Rollback                                     <<<
:  |  <--QUIT                                      <<<
:  | -*OTHERWISE                                  <<<
:  |  Commit                                       <<<
:  '-ENDCASE                                      <<<
'--                                               <<<
```

For more information about commitment control, see this module, in the chapter, "Modifying Function Options."

Compute

The *COMPUTE built-in function enables you to define a complex arithmetic expression using the following mathematical operators on a single compute expression line.

Operator	Operation	Definition
+	*ADD	addition
–	*SUB	subtraction
*	*MULT	multiplication
/	*DIV	division
\	*MODULO	modulo

For more information on these operations, see, Understanding Built-In Functions *ADD, *SUB, *MULT, *DIV, and *MODULO subtopics earlier in this chapter.

There is one output parameter, the *Result field of the object type FLD, associated with this function type. It contains the result of the computation expression.

Note: You can define several additional parameters as needed by the details of the compute statement.

By default, intermediate results for each operation are contained in *Synon (17,7) Work fields. You can override the work fields with any valid field in the action diagram to contain intermediate results.

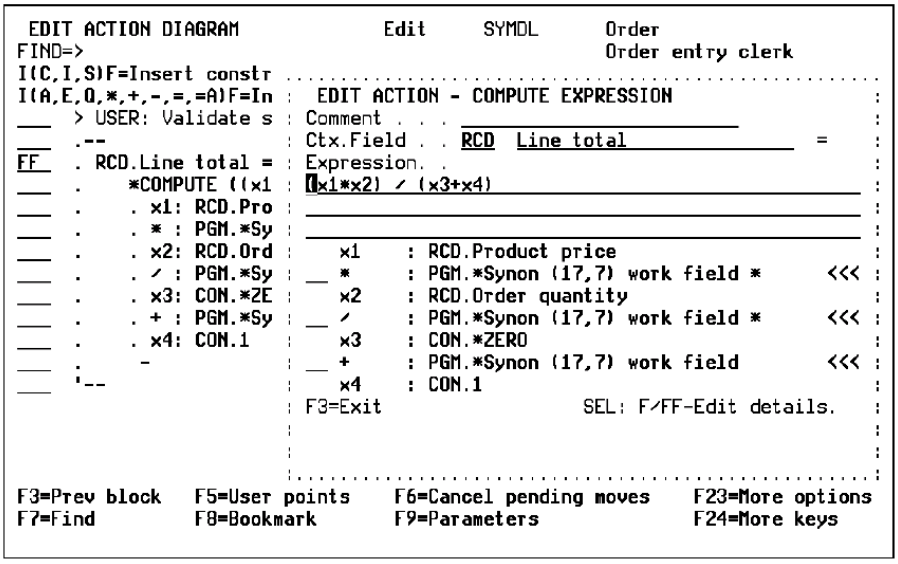
Note: The precision of the intermediate result fields affects the overall precision of the *COMPUTE expression. For example, the default fields are defined with a length of 7 decimal digits. Any rounding you specified for a multiplication or division operation occurs only in the following cases:

- The intermediate result has more than 7 decimal digits
- The operation is last in the *COMPUTE expression and the length of the final result field has fewer decimal digits than the calculation requires

To force rounding, ensure that intermediate and final result fields have the appropriate number of decimal digits.

Defining a Compute Expression

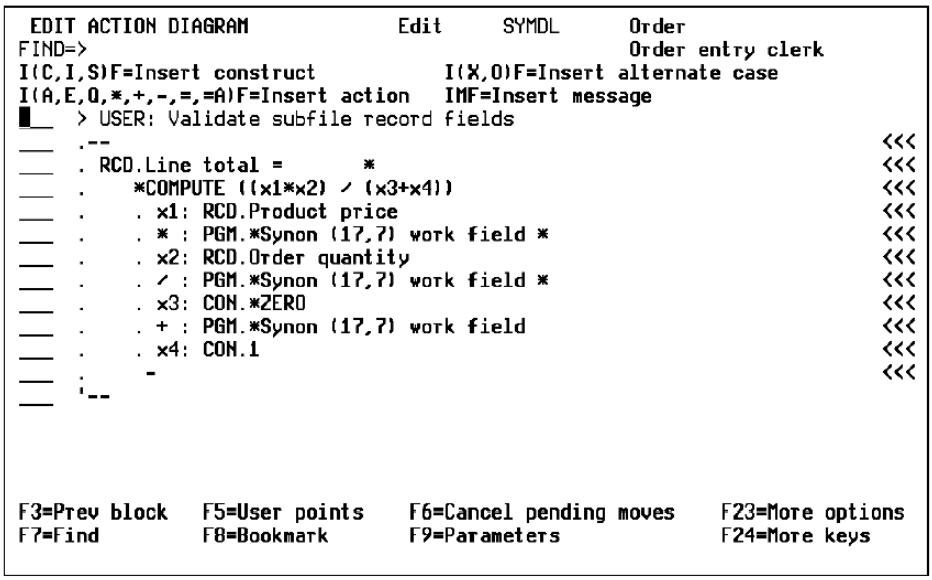
Enter the built-in function *COMPUTE or press F7 on the Edit Action Function Details panel to convert an existing arithmetic built-in function to a compute expression. This action causes the Edit Action-Compute Expression panel to display.



On the Edit Action - Compute Expression panel the *Compute expression appears as an input-capable character field. Specify the arithmetic expression using the correct mathematical formula.

CA 2E writes the correct field contexts and expressions under the compute line. You must then use the F or FF line commands to edit the function name or the function details associated with each pair of terms and operations in the expression.

For more information on the line commands, see the Understanding the Action Diagram Editor topic in this chapter.



Note: Encapsulating the compute expression in a derived field allows the expression to be easily re-used in several functions.

Concatenation

The *CONCAT built-in function provides the means of joining or concatenating two discrete strings of data into a single string.

For more information on concatenating numeric data without conversion to character data, see this topic, Convert Variable.

There are four parameters for this function type.

- Three input parameters: a character field, *String 1 of usage VRY which is the first string to be joined; a character string, *String 2 also of usage VRY which is the second string to be joined to *String 1; and a numeric field, *Number of blank, that determines the number of blanks between the two strings.

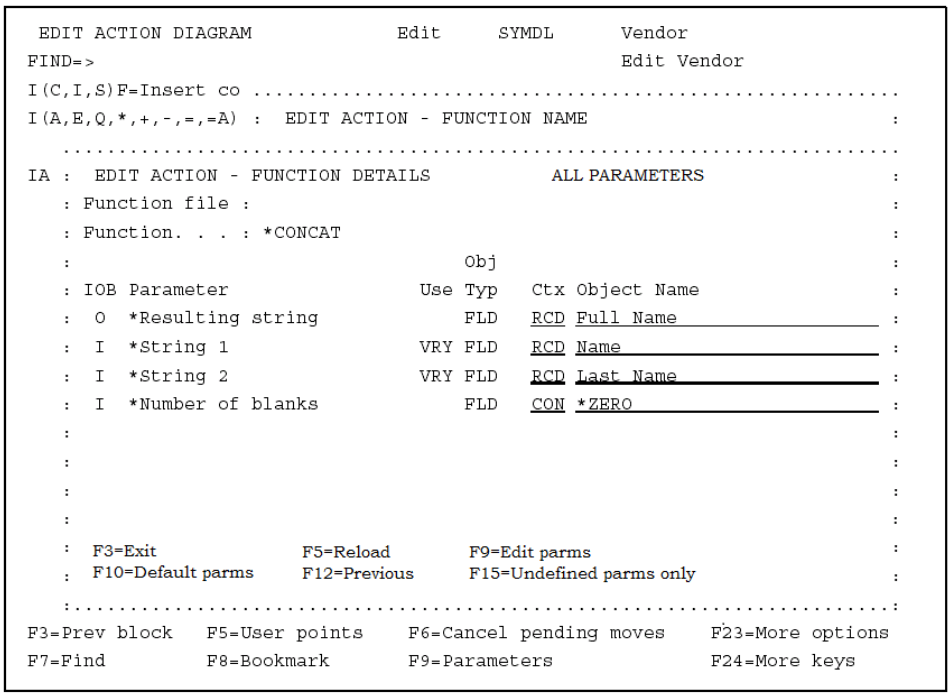
  The following three conditions are supplied with the *Number of blanks parameter.

Condition	Result
*None	No blanks between the two strings
*One	Single blank between the two strings
*All	Retain all trailing blanks of the first string

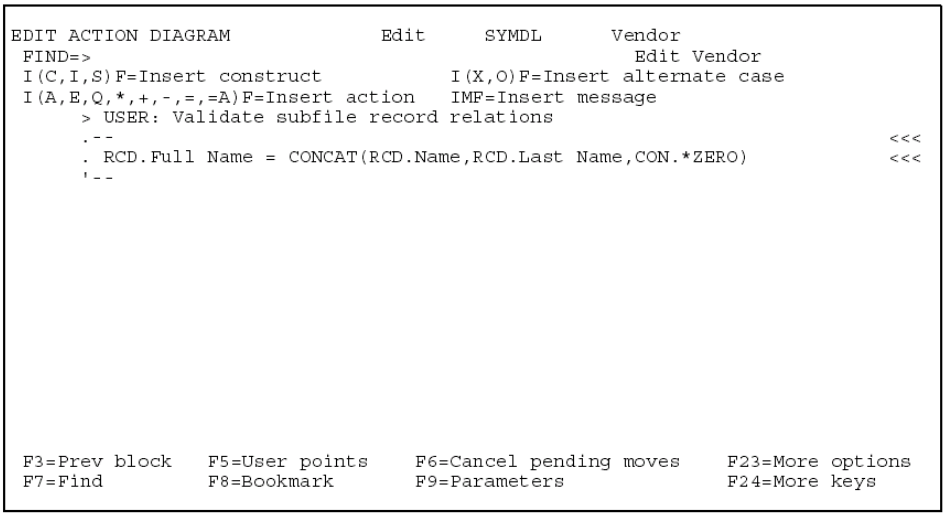
- One output parameter, which is the *Resulting string or the string that comprises the two joined strings

CA 2E implements the *CONCAT built-in function as an RPG CAT statement. In COBOL CA 2E implements the *CONCAT function as a STRING statement.

The following example shows a concatenation function that concatenates the fields Name and Last Name and does not put any blanks between the two strings.



The following example shows how the concatenation appears in the Action Diagram:



Convert Variable

The *CVTVAR built-in function specifies that the value of one CA 2E field is to be moved to another field of a different type; that is, the two fields do not have to be of the same domain. CA 2E converts the field values according to the assignment rules of the HLL language in which you create the function.

An example of the use of this function might be to move a numeric code, stored in a CDE field, into a NBR field.

Note: To convert among date (DTE, DT#, TS#), time (TME, TM#), and number (NBR) data types, use the *MOVE built-in function instead.

You can also use the *CVTVAR built-in function with the ELM context to move data between a field and a data structure. In CA 2E a data structure is equivalent to a single element array. This provides a method for decomposition or (re)composition of field data in a single operation.

For example, you can use this technique to compose a complex data string into a single parameter required by a system API (Application Interface Program) or a third party application. Conversely, you can use this technique to decompose and recompose a telephone number or postal code.

For more information on the ELM context field, see Understanding Contexts in this chapter.

There are two parameters associated with the *CVTVAR function type:

- **One input parameter (*FACTOR2)**—The field of any attribute or domain that is to be moved.

  Note: Fields with a context of CND or CON are not appropriate for the *CVTVAR input parameter.

- **One output parameter**—The *Result field, also of any attribute or domain, into which the field is to be moved.

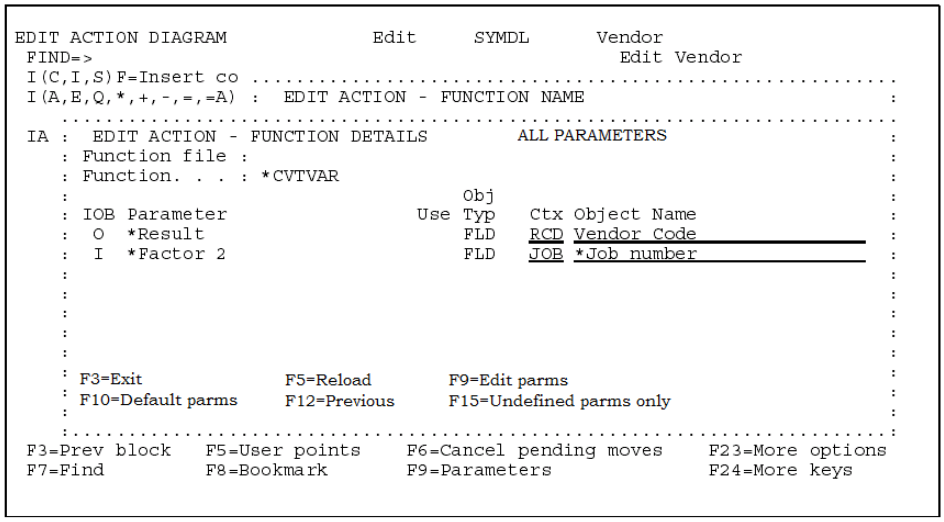
By default, CA 2E implements the *CVTVAR built-in function as an RPG MOVEL statement when moving from a numeric field into an alphanumeric field. CA 2E uses an RPG MOVE statement when moving from an alphanumeric field into a numeric field. CA 2E implements the *CVTVAR function as a COBOL MOVE statement.

If the *Result field is longer than the moved field the result field is blanked out or converted to zeroes prior to the move. Any excess characters are converted to blanks or zeros.

Note: If you move an alphanumeric field to a numeric field, COBOL does not convert spaces to zeroes. This can cause decimal data errors.

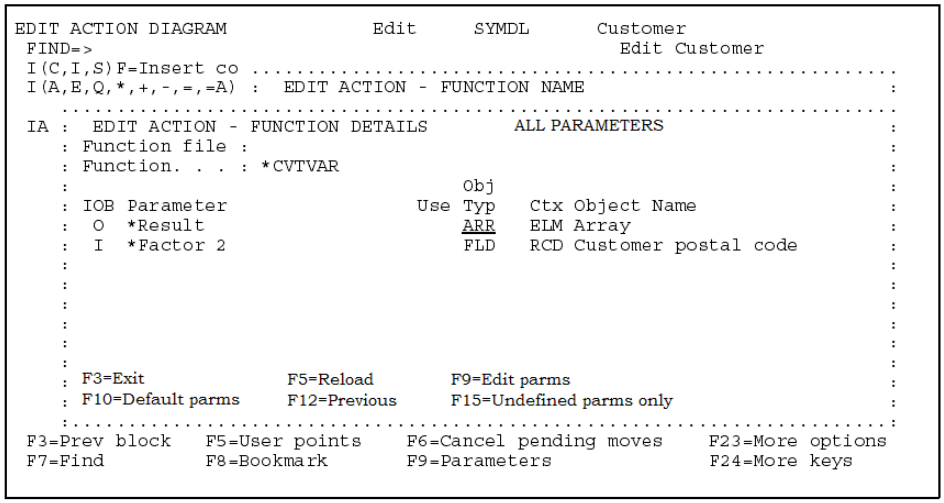
Example 1

This example shows how to move a number into a code field.



Example 2

This example shows how to decompose a field (Customer postal code) into a structure defined by an array.



Date Details

The *DATE DETAILS built-in function returns information about a given date; for example, day of week, length of month, or whether it is in a leap year. You specify the kind of information you need using the *Date detail type parameter.

Note: You should check the return code set by this function in order to catch any errors encountered.

There are seven parameters for this function type:

- Six input parameters:
  - *Date is the date for which information is to be returned. A field of type NBR is interpreted as the number of days since January 1, 1801 (day one).
  - *Date detail type specifies the kind of information returned for the *Date field. See the table at the end of this topic.
  - *Excluded days of week specifies days that are normally excluded from the operation; for example, weekends.
  - *Date List name specifies the name of an existing date list. Date lists let you override selection rules set by the *Excluded days of week parameter for particular dates; for example, holidays. The specified date list needs to be in the *Date Lists array when the built-in function executes.
  - *Date List autoload specifies whether the function automatically loads the date list into the *Date Lists array when it executes.
  - *Select days/dates lets you reverse the selection determined by the *Excluded days of week and *Date List name parameters. The default is to provide date details for included days only.
- One output parameter, *Date detail. This field contains the requested information for the input date. Its meaning is determined by the *Date Detail Type.

For more information on the selection input parameters, see the Selection Parameters for Date Built-In Functions subtopic later in the Date Details topic.

The possible values for the *Date detail type and the effect of each on the meaning of the output field are summarized in the following table.

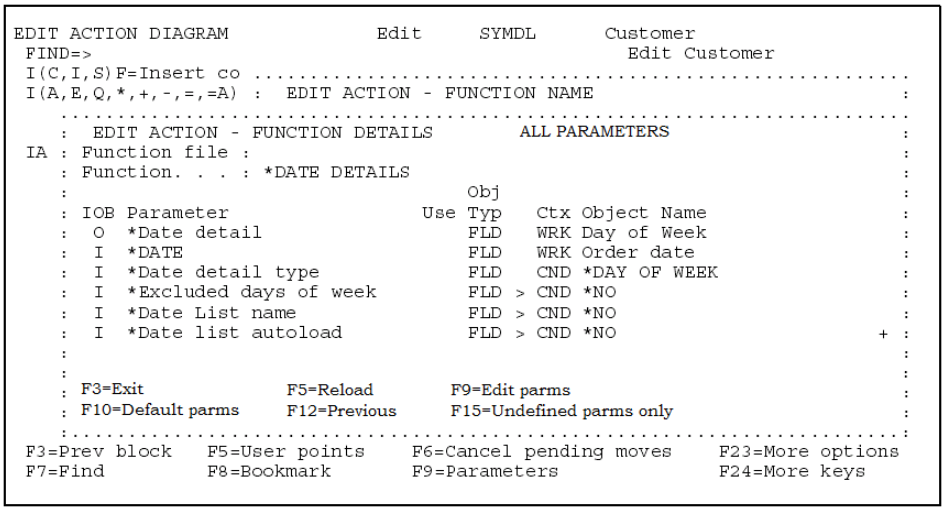
*Date Detail Type Values	Effect on the *Date Detail Parameter
*ABSOLUTE DAY	The result is the number of days that have elapsed since January 1, 1801 (the day one) for the given date.
*DAY OF YEAR	The result is an integer from 1 to 366, specifying the number of selected days that have elapsed since the beginning of the given year.

*Date Detail Type Values	Effect on the *Date Detail Parameter
*DAY OF MONTH	The result is an integer from 1 to 31, specifying the number of selected days that have elapsed since the beginning of the given month.
*DAY OF WEEK	The result is an integer from 1 to 7, specifying the number of selected days that have elapsed since the beginning of the given week. The days of the week are numbered sequentially beginning with 1=Monday.
*SELECTED?	The result specifies whether the given date was selected. See the *Select Days/ Dates parameter. 1=the date was selected. 0=the date was not selected.
*MONTH	The result is an integer from 1 to 12, specifying the month of the given date.
*MONTH LENGTH	The result is an integer specifying the number of selected days in the month for the given date.
*YEAR	The result is the year of the given date, in the format YYYY.
*LEAP YEAR?	The result specifies whether the given date is in a leap year. 1=the date is in a leap year. 0=the date is *not* in a leap year.
*YEAR LENGTH	The result is the number of selected days in the year for the given date.

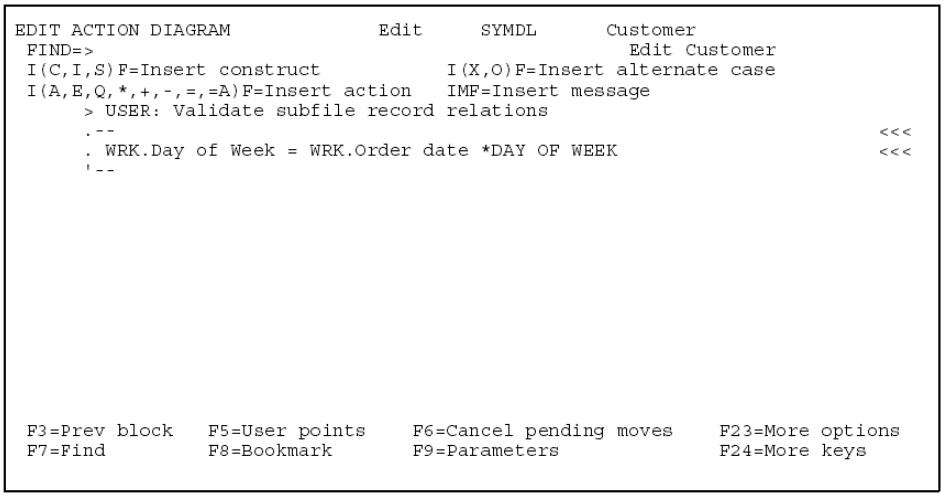
Note: The result in *Date detail reflects only *selected* days unless you specify *ABSOLUTE DAY, *MONTH, *YEAR, or *LEAP YEAR for *Detail type.

Example

Suppose you want to calculate the day of the week an order was placed. For example, if the order date was, August 4, 1994, the result is Thursday. Following are parameter specifications for the *DATE DETAILS built-in function that produce this result. Scroll to view the *Selected days/dates parameter.



To insert this *DATE DETAILS built-in function into the action diagram, press Enter.



Selection Parameters for Date Built-In Functions

This topic gives details about the selection parameters for the *DATE DETAILS, *DATE INCREMENT, and *DURATION built-in functions. It also discusses the related *Date list autoload parameter. The selection parameters are

- *Excluded days of week
- *Date List name
- *Select days/dates

Note: For the *DATE INCREMENT and *DURATION built-in functions, you can specify a value other than *NO or NONE for the selection parameters only if the *Duration type parameter is *DAYS.

*Excluded Days Of Week

This parameter is a condition field or derived field of type STS. Use this parameter to specify days that are normally to be excluded from an operation; for example, weekends or days not worked by part-time employees.

Each value you specify for this parameter consists of seven digits. Each digit can be 1 or 0 and corresponds to a day of the week beginning with Monday. A 1 indicates that the day is to be included in the operation; a 0 indicates that the day is to be excluded.

The possible values are shown in the following table.

*Excluded Days of Week Values	Description and Examples
*NO	Include all days of the week. The value is 1111111.
*SUNDAY	Exclude Sundays. The value is 1111110.
*SATURDAY	Exclude Saturdays. The value is 1111101.
*SATURDAY, SUNDAY	Exclude Saturdays and Sundays. The value is 1111100.
User-defined	You define which days of the week to include and exclude; for example, if your department works Tuesday through Saturday, define a condition with value 0111110.

You can modify the selection rule set by this parameter using the other two selection parameters.

- Use *Date List name to exclude or include particular dates.
- Specify *EXCLUDE for *Select days/dates to reverse the effect of the selection; in other words, to select excluded days.
- *Date List Name

This parameter specifies the name of an existing date list.

Date lists let you override the selection rules set by the *Excluded days of week parameter for particular dates; for example, holidays. A date list consists of a unique name, a list of dates, and a 1 or 0 for each date to indicate whether to include or exclude the date. To use a date list, specify its name on the *Date List name parameter.

Note: You can specify dates for different years on the same date list.

The default for *Date List name is NONE; in other words, no date list is specified. This value is required in the following cases.

- The *Duration type parameter is other than *DAYS on the *DATE INCREMENT and *DURATION built-in functions.
- The *Date detail type parameter is *ABSOLUTE DAY, *MONTH, *YEAR, or *LEAP YEAR on the *DATE DETAILS built-in function.

When a date built-in function uses a date list executes, it expects to find the specified *Date List name in the *Date Lists array. This array and the function needed to create it are shipped with CA 2E. Each element of the array is comprised of the following fields:

Field Name	Type	
*Date List Name	VNM	Key
*Date absolute day	NBR	Key
*Date flag	STS	Atr

For more information on arrays, see Building Access Paths, in the chapter "Defining Arrays."

To load a date list into the *Date Lists array you can do one of the following:

- Write instructions in the action diagram to load the information into the array before executing the built-in function.
- Specify *YES for the *Date list autoload parameter. The *Date list autoload parameter determines whether the function is to automatically load the specified date list into the *Date Lists array, if it does not find the name in the array. The possible values are
  - *YES: The specified *Date List name is automatically loaded from the *Date List Detail file into the *Date Lists array when the built-in function executes. You can specify *YES only when *Date List name is other than NONE.
  - *NO: You need to provide instructions in the action diagram to load the date list into the *Date Lists array before executing the built-in function.

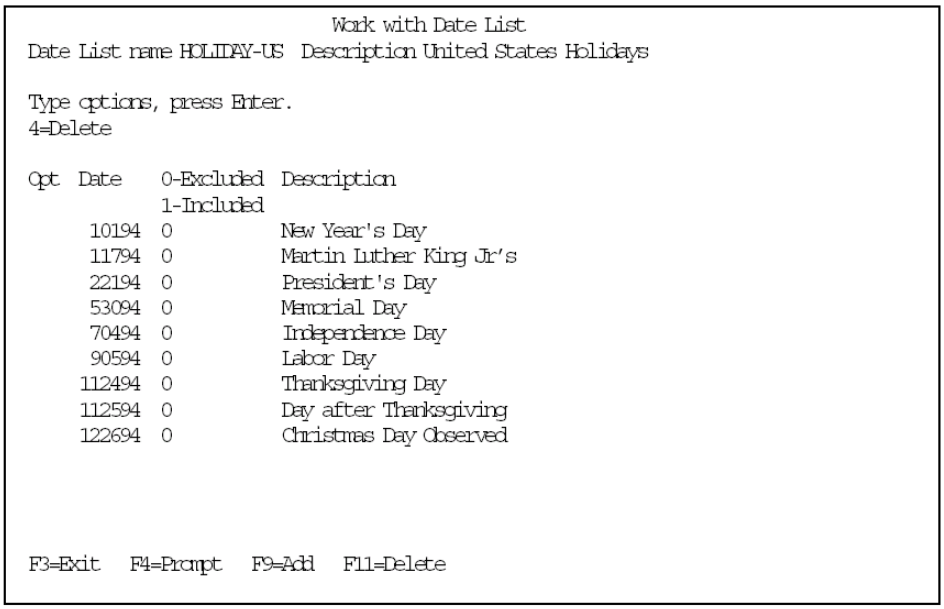
To create date lists for use with the *Date list autoload capability, you can use the Work with Date List function that is shipped with CA 2E in the *Date List Header file. The information you enter is stored in the *Date List Header and *Date List Detail files.

The *Date List Header and *Date List Detail physical and logical files are supplied in the generation library in addition to being defined in the model. If you want to maintain these files, you need to regenerate and compile them.

Notes:

- You need to generate and compile the Work with Date List function into your generation library. You invoke it using its implementation name.
- Before you can compile functions with *Date list autoload set to *YES, you need to generate and compile the PHY, RTV, and UPD access paths for the *Date List Detail file.

Suppose Company ABC is closed for business on certain holidays. The following example shows a date list, created using the Work with Date List function that excludes those holidays.



You can create a similar date list to *include* days that are not considered normal business days. For example, suppose the employees of Company ABC are required to work on a Saturday for the company inventory. The date list would contain an entry like the following.

```
70994 1      Company Inventory Day
```

You can reverse the effect of the selection rule set by a date list by specifying *EXCLUDE for the *Select days/dates parameter; in other words, you can select excluded days instead of included days.

*Select Days/Dates

This parameter lets you reverse the selection set by the *Excluded days of week and *Date List name parameters. Normally, only included days are selected and considered by a date built-in function. This parameter lets you select excluded days instead.

The possible values for this parameter are shown in the following table.

*Select Days/ Dates Values	Effect on the Output of the Date Built-In Function
*INCLUDED	Select only days that are either Flagged as included on the *Date List. 2. Not excluded by *Excluded days of week and not flagged as excluded on the *Date List. This is the default.

*Select Days/ Dates Values	Effect on the Output of the Date Built-In Function
*EXCLUDED	Select only days that are either Flagged as excluded on the *Date List. Excluded by *Excluded days of week and not flagged as included on the *Date List. This lets you reverse the default selection.
*NO	The default is automatically changed to *NO when the built-in function does not require selection; for example, when *Duration type is not *DAYS or when *Date details is *ABSOLUTE DAY. If the built-in function requires selection, you cannot specify *NO.

Date Increment

The *DATE INCREMENT built-in function lets you add a quantity to a given date. You specify the kind of quantity to add using the *Duration type parameter. Note that you should check the return code set by this function in order to catch any errors encountered. This function is the converse of the *DURATION function.

The *DATE INCREMENT built-in function performs the operation:

```
*Date1 = *Date2 + *Duration
```

There are eight parameters for this function type:

- Seven input parameters
  - *Date2 specifies the beginning date. If it is of type NBR, it is interpreted as the number of days since January 1, 1801 (day one).
  - *Duration specifies the quantity to be added to the beginning date. Its meaning is determined by the value of *Duration type.
  - *Duration type specifies the meaning of the quantity to be added to the beginning date. See the table at the end of this topic.
  - *Excluded days of week specify days that are normally not to be included in the sum.
  - *Date List name specifies the name of an existing date list. Date lists let you override selection rules set by the *Excluded days of week parameter for particular dates. The specified date list needs to be in the *Date Lists array when the built-in function executes.
  - *Date list autoload determines whether the function is to automatically load the specified date list into the *Date Lists array when the function is executed.
  - *Select days/dates lets you reverse the date selection determined by the *Date List name and *Excluded days of week parameters. The default is to select included dates.
- One output parameter, *Date1, which specifies the result date. If it is of type NBR, it is interpreted as the number of days since January 1, 1801 (day one).

For more information on the selection input parameters briefly described here, see the Selection Parameters for Date Built-In Functions topic in the *DATE DETAILS built-in function description.

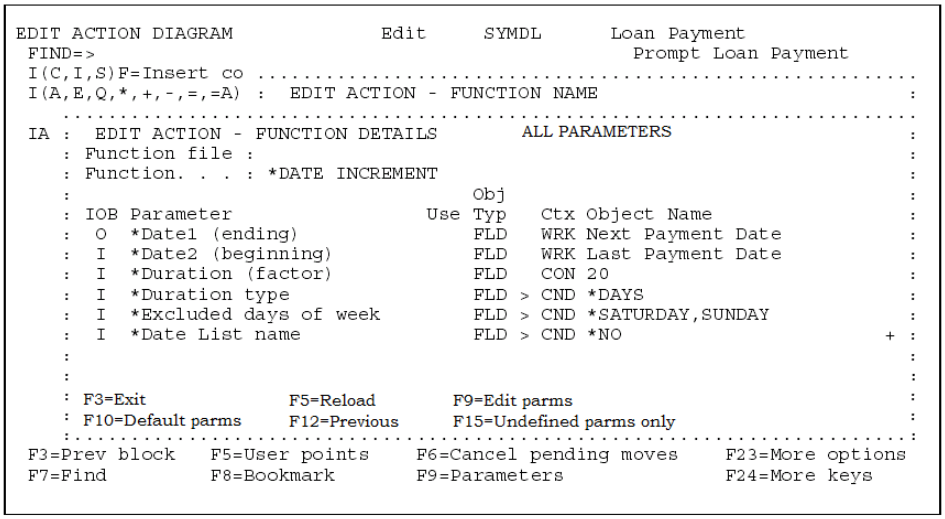
The possible values for *Duration type and the effect each has on the meaning of the *Duration parameter are shown in the following table.

*Duration Type Values	Effect on the *Duration Parameter

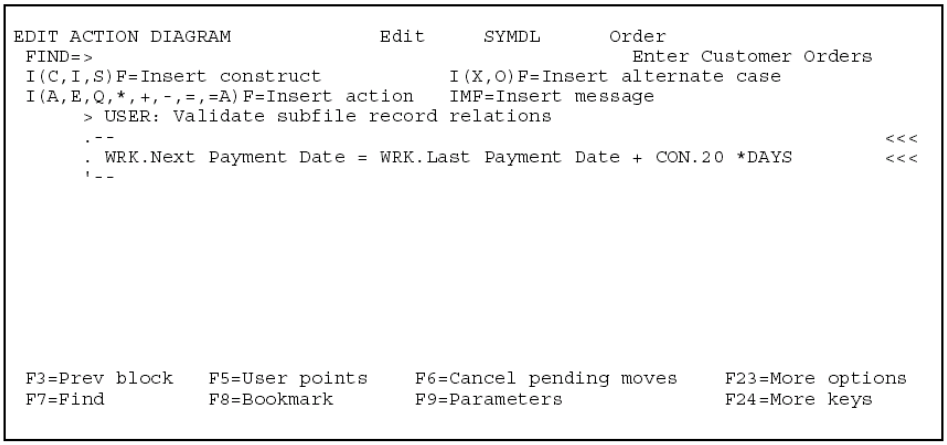
*Duration Type Values	Effect on the *Duration Parameter
*DAYS	Number of selected days to add to the specified date (*Date2). This is the default.
*MONTHS	Number of full months to add to the specified date (*Date2). Partial months are ignored.
*YEARS	Number of full years to add to the specified date (*Date2). Partial years are ignored.
*YYMM	Number of years and full months in YYMM format to add to the specified date (*Date2); partial months are ignored. For example, 1011 means 10 years and 11 months; 1100 means exactly 11 years.
*YYMMDD	Number of years, months, and days in YYMMDD format to add to the specified date (*Date2). For example, 100923 means 10 years, 9 months, and 23 days; 110000 means exactly 11 years.

Example

Suppose you want to calculate the next interest payment date for a loan where an interest payment is due every 20 days, not including weekends. For example, if the last payment date was, August 4, 1994, the next interest payment is due, September 1, 1994. Following are parameter specifications for the *DATE INCREMENT built-in function that produce this result. Scroll to view the *Date list autoload and *Selected days/dates parameters.



To insert this *DATE INCREMENT built-in function into the action diagram, press Enter.



For more information on *DATE INCREMENT, see Calculation Assumptions and Examples for Date Built-In Functions at the end of this section.

Divide

The *DIV built-in function specifies an arithmetic division of one field by another. You can specify the shipped field *Rounded to determine whether the result of the division is half-adjusted or not; CA 2E provides two conditions for this purpose.

There are four parameters for this function type:

- Three input parameters which are the dividend (*FACTOR1), the divisor (*FACTOR2), and the *Rounded field.
- One output parameter which is the *Result field containing the result of the division.

*FACTOR1, *FACTOR2, and the *Result field must all be numeric field types.

CA 2E implements the *DIV built-in function as an RPG DIV statement and as a COBOL DIVIDE statement.

Example

This is an example of how to use the *DIV built-in function.

```
> USER: Process detail record
.--
:—CTL. Average value = WRK.Total val/WRK.Total <<<
no.
‘--
```

Divide with Remainder

The *DIV WITH REMAINDER built-in function specifies an arithmetic division of two fields with the remainder being stored in an additional field.

There are four parameters for this function type:

- Two input parameters, which are the dividend (*FACTOR1) and the divisor (*FACTOR2).
- Two output parameters which are the *Remainder field and the *Result field containing the result of the division.

All parameters must be numeric field types.

CA 2E implements the *DIV WITH REMAINDER built-in function as an RPG MVR statement, and as a COBOL DIVIDE statement followed by a COBOL MOVE statement.

Duration

The *DURATION built-in function calculates the elapsed time between a beginning date and an ending date.

Note: You should check the return code set by this function in order to catch any errors encountered. This function is the converse of the *DATE INCREMENT built-in function.

The *DURATION built-in function performs the operation:

```
*Duration = *Date1 – *Date2
```

The result is positive if *Date1 is after *Date2; it is negative if *Date1 is before *Date2.

There are eight parameters for this function type:

- Seven input parameters
  - *Date2 and *Date1 specify the beginning and ending dates, respectively. When either date is of type NBR, it is interpreted as the number of days since January 1, 1801 (day one).
  - *Duration type specifies the meaning of the result of the operation. See the table at the end of this topic.
  - *Excluded days of week specifies the days to exclude from the operation; for example, weekends or days not worked by part-time employees.
  - *Date List name specifies the name of an existing date list. Date lists let you override normal selection rules set by the *Excluded days of week parameter for particular dates. The specified date list needs to be in the *Date Lists array when the built-in function executes.
  - *Date list autoload determines whether the function automatically loads the specified date list into the *Date Lists array when the function is executed.
  - *Select days/dates lets you reverse the selection determined by the *Excluded days of week and *Date List name parameters. The default is to select included dates.
- One output parameter, *Duration. The meaning of this parameter is determined by the value of *Duration type.

Note: The *Date list, *Excluded days of week, and *Select days/dates parameters affect only days/dates after the beginning date. If the ending date is before the beginning date, these parameters affect only days/dates after the ending date.

For more information on the selection input parameters, see the Selection Parameters for Date Built-In Functions topic in the *DATE DETAILS built-in function description.

The possible values for *Duration type and the effect each has on the meaning of *Duration are shown in the following table.

*Duration Type Values	Effect on the *Duration Parameter
*YEARS	The result is given as a number of full years; partial years are ignored.
*MONTHS	The result is given as a number of full months; partial months are ignored.
*YYMM	The result is given as a number of years and full months in YYMM format; partial months are ignored. For example, 1011 means 10 years and 11 months; 1100 means exactly 11 years.
*YYMMDD	The result is given as a number of years, months, and days in YYMMDD format. For example, 100923 means 10 years, 9 months, and 23 days; 110000 means exactly 11 years.
*DAYS	The result is the number of selected days. This is the default.

Elapsed Time

The *ELAPSED TIME built-in function calculates the elapsed time between a beginning time and an ending time. It is the converse of the *TIME INCREMENT built-in function.

The *ELAPSED TIME built-in function performs the operation:

```
*Elapsed Time = *Time1 – *Time2
```

The result is positive if *Time1 is after *Time2; it is negative if *Time1 is before *Time2.

There are four parameters for this function type:

- Three input parameters
  - *Time2 and *Time1 specify the beginning and ending times, respectively. When either time is of type NBR, it is interpreted as the elapsed time since 0 a.m.
  - *Time unit specifies the meaning of the *Elapsed time output parameter.
- One output parameter, *Elapsed time. The meaning of this parameter is determined by the value of *Time unit.

The valid values for *Time unit and the effect each has on the meaning of *Elapsed time are shown in the following table.

*Time Unit Values	Effect on the *Elapsed Time Parameter
*SECONDS	The result is given as an integer specifying the number of elapsed seconds.

*Time Unit Values	Effect on the *Elapsed Time Parameter
*MINUTES	The result is given as the number of elapsed minutes; partial minutes are ignored.
*HOURS	The result is given as the number of elapsed hours; partial hours are ignored.
*HHMM	The result is given as the number of elapsed hours and minutes in HHMM format.
*HHMMSS	The result is given as the number of elapsed hours, minutes, and seconds in HHMMSS format.

Exit Program

The *EXIT PROGRAM built-in function specifies an exit from a program.

The only parameter for this built-in function is the *Return code. You can use the *Return code parameter to inform the calling program of the circumstances under which the program was exited.

CA 2E supplies the *Return code field from the PGM context as an input parameter, by default. You can also supply alternate conditions to the *Return code field, other than those supplied by default, such as *Record does not exist.

CA 2E implements the *EXIT PROGRAM built-in function as a call to a CA 2E supplied exit subroutine, ZYEXPG. This issues an *EXIT PROGRAM in COBOL or in RPG. If closedown program is specified, RPG also sets on the LR indicator.

Example

The following is an example of the *EXIT PROGRAM built-in function.

```
>  Fast exit                                              <<<
.—CASE:                                                   <<<
|-CTL. *CMD key is CF13                                   <<<
|  *Exit program — return code CND. *User QUIT request    <<<
‘--ENDCASE
```

Modulo

The *MODULO built-in function specifies the remainder of a division of two fields. The *MODULO function provides more control over the remainder precision and is a single value field as opposed to the *DIV With Remainder, which returns two values. This allows this function to be used in *COMPUTE functions.

There are four parameters for this function type:

- Three input parameters which are the divisor, the dividend of the division operation, and a *Quotient definition field. The latter specifies the field domain to be used in defining the intermediate work field generated by *MODULO to contain the quotient of the intermediate division operation.
- One output parameter, a *Result field that contains the result of the entire *MODULO function.

Note: The final result of the *MODULO operation depends greatly on the field domain defined for both the *Quotient definition field and the *Result field. For example, suppose you want to calculate the modulo for the following expression:

```
5.30 / 2.10 = 2.5238
```

The following table shows three different modulo values (*Result) for this operation due to the field length defined for the *Quotient definition field and the *Result field.

Length of *Quotient Definition Field	Length of *Result Field	Quotient (Work Field VValue)	Modulo (*Result value)
4.0	6.4	2	1.1
4.2	6.4	2.52	.0080
4.2	6.2	2.52	.00

You can use the *MODULO built-in function as a sub-function to the *COMPUTE built-in function, thereby determining the remainder of a division operation within the compute expression.

CA 2E implements the *MODULO built-in function using similar code to *DIV (with remainder).

Example

This is an example of the *MODULO built-in function.

```
>USER: Process detail record
.--
:  WRK.MODULO Field = RCD.EXT Price\RCD.Quantity      <<<
'--
```

Move

The *MOVE built-in function specifies that the value of one field is to be moved to another.

There are two parameters for this function type:

- One input parameter, which is the field that is to be moved (*FACTOR2).
- One output parameter, which is the *Result field into which the field is moved.

For all but date and time fields, the moved and result fields must either both be numeric or both be alphanumeric field types. Refer to Considerations for Date and Time Field Types at the end of this topic for details about date and time conversions.

Note: If CND is specified for the context of *FACTOR2 and the field is a status field, the condition must be a VAL condition. If the field is not a status field and CND is specified, the condition must be a CMP condition with an operation of EQ.

CA 2E implements the *MOVE built-in function as an RPG Z-ADD statement for numeric fields and as a MOVEL statement for alphanumeric fields; in COBOL CA 2E implements the *MOVE function as a MOVE statement.

Example

This is an example of the *MOVE built-in function.

```
> USER: Process subfile record
.--
:   Execute another function                  <<<
:   PGM. *Reload subfile = CND. *Yes          <<<
'--
```

Move Array

The *MOVE ARRAY built-in function lets you move multiple instances of an array. To specify an array subfield, specify the Array Subfield name, Array Name, and Array Index (Element Number).

Use *MOVE ARRAY in the following ways:

- Move the value of one array subfield into another array subfield, either in the same array or a different array.
- Move the value of an array subfield into a field in a non-array context, for example, WRK or LCL.
- Move the value of a field in a non-array context or a constant value or a valid condition into an array subfield.

Note: In all these cases, the special value **ALL* can be used in place of a field name. *ALL pertains to all fields in the specified array.

For more information and examples, see the *MOVE ARRAY Examples (see page 473).

When using the *MOVE ARRAY function, the array must be a multiple-instance array, using the ARR context, or a parameter context where the parameter is defined as a multiple-instance array parameter.

For a working scenario using the *MOVE ARRAY built-in function, see the Appendix How to Create a Deployable Web Service Using a Multiple-instance Array (see page 743).

Move Array Parameters

The *MOVE ARRAY built-in function uses the following parameters.

Note: The first three parameters define the target field, and the last three parameters define the source field:

***Result**

The target field or the special value *ALL

***Array**

The array in which the target field exists (if it is an array subfield)

***Array index**

The index number which specifies the element of the array in which the target field exists (if it is an array subfield)

***Factor 2**

The source field or the special value *ALL

***Array**

The array in which the source field exists (if it is an array subfield)

***Array index**

The index number that specifies the element of the array where the source field exists (if it is an array subfield)

Within the Action Diagram, the syntax of each group of three fields is as follows (when *Array and *Array index are specified):

```
array-context.array(array-index-context.array-index).array-subfield
```

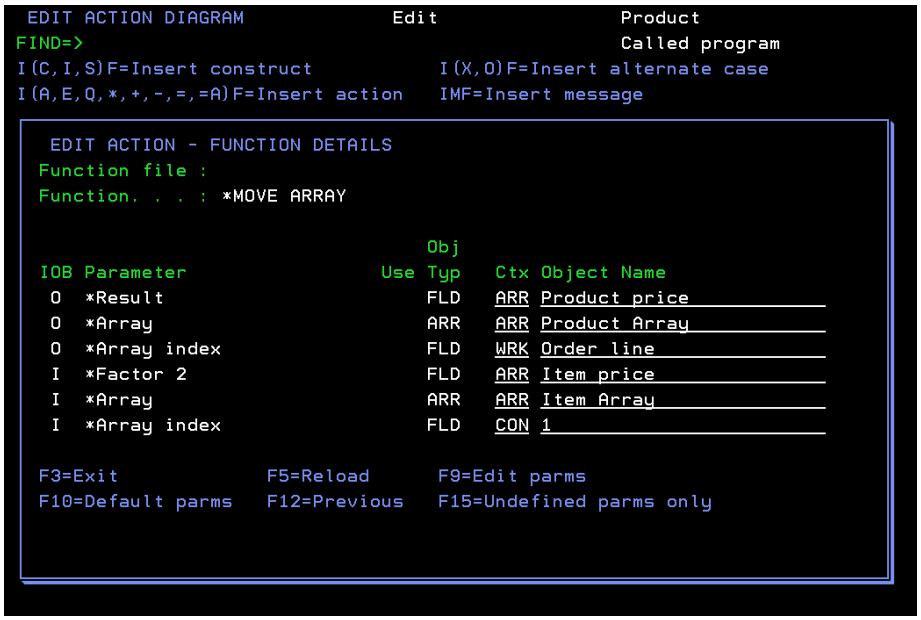
Unlike most function calls, some of the parameters, and their contexts, to the *MOVE ARRAY built-in function can be blank, as shown in the following examples.

Move Array Examples

Use *MOVE ARRAY in the following situations:

- Move an array subfield within a specified element of an array into another array subfield, either in the same array or a different array.

  In this example, the Product price subfield in the element of the Product Array, which the current value of the Order line field specifies, is set to the value held in the Item price subfield in the first element of the Item Array:

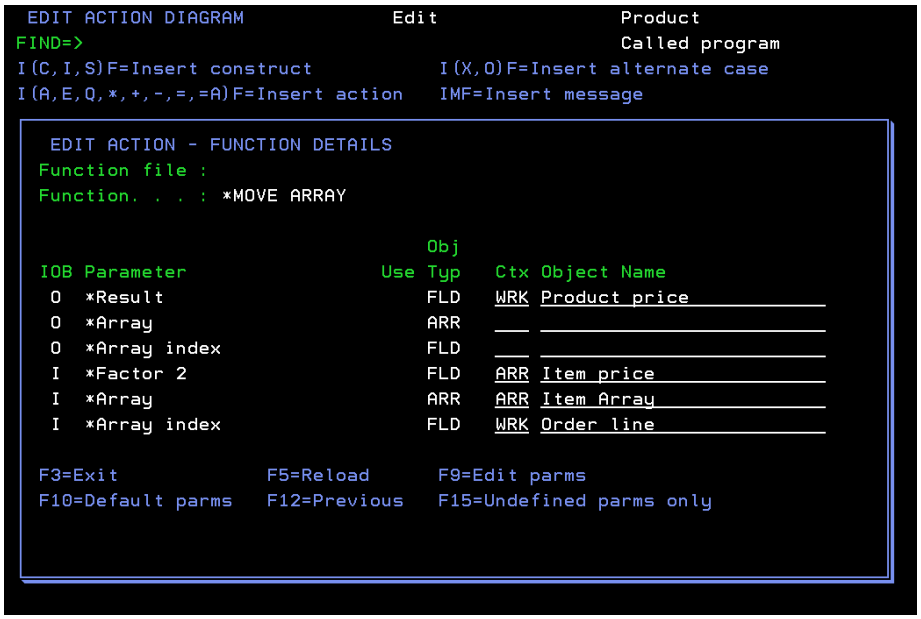


  This Action Diagram statement displays as:

```
ARR.Product Array(WRK.Order line).Product price =
ARR.Item Array(CON.1).Item price
```

- Move an array subfield into a field in a non-array context, for example, a field in the WRK context.

  In this example, the Product price field in the WRK context is set to the value held in the Item price subfield in the element of the Item Array specified by the current value of the Order line field:

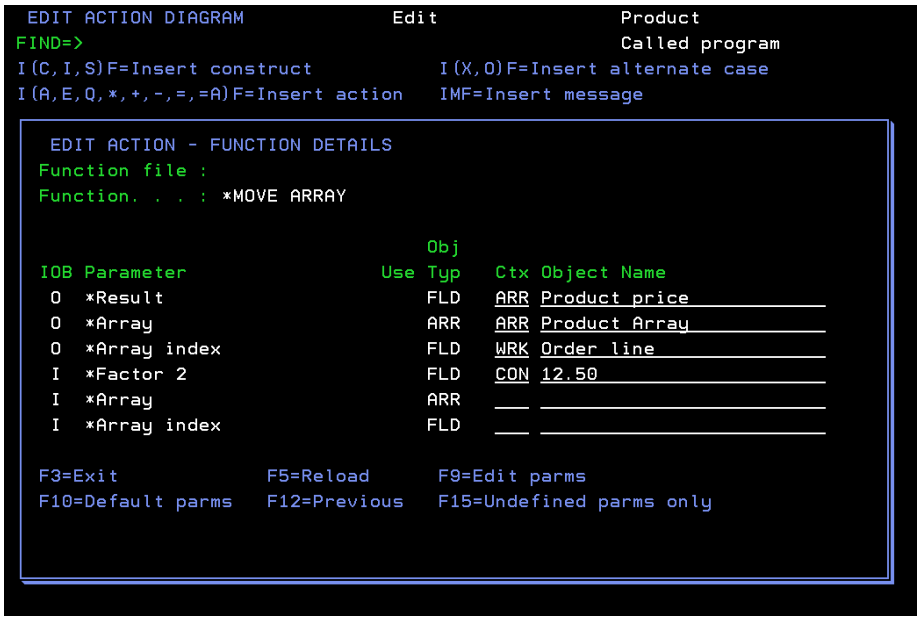


  This Action Diagram statement displays as follows:

```
WRK.Product price = ARR.Item Array(WRK.Order line).Item price
```

- Move a field or value (including conditions and constants) in a non-array context into an array subfield.

  In this example, the Product price subfield in the element of the Product Array specified by the current value of the Order line field is set to a value of 12.50:



  This Action Diagram statement displays as follows:

```
ARR.Product Array(WRK.Order line).Product price = CON.12.50
```

Move Array Usage

If you specify the ARR context, in addition to the code required for the *MOVE ARRAY function, CA 2E generates code to define the required array structures. Therefore, you can define these arrays automatically by using the *MOVE ARRAY function. When you generate a *MOVE ARRAY statement, CA 2E generates additional code that checks for array indexing errors.

Multiple instance arrays in 2E are 1-based; the first element in an array is element 1. You cannot specify a constant value (CON context) less than 1 or greater than the maximum number of elements in the array in the Action Diagram Editor. However, if you specify a runtime field value less than 1 or greater than the maximum number of elements in the array in the *Array index field, you receive an error.

If an array indexing error occurs, the PGM.*Return code field is set to a condition value of '*Array index error', which corresponds to the Y2U0068 message in the Y2USRMSG message file. This error can be monitored for as in the following example:

```
WRK.Product price = ARR.Item Array(WRK.Order line).Item price
.-CASE
¦-PGM.*Return code is *Array index error
¦ <-- *QUIT
'-ENDCASE
```

If *Result or *Factor 2 is not an array subfield, the following restrictions apply to that field:

- The related *Array and *Array index fields must be blank.
- The context specified must be one that would be valid in a *MOVE statement.
  - The valid contexts for *Result are PGM, LCL, WRK and NLL (including any valid parameter context, if the function has an appropriate output parameter).
  - The valid contexts for *Factor 2 are PGM, JOB, LCL, WRK, CND and CON. This context includes any valid parameter context, if the function has an appropriate input parameter.

If *Result or *Factor 2 is an array subfield then the following restrictions apply to that field.

- The related *Array and *Array index fields cannot be blank.
  - The *Array index can be a positive integer constant with a value greater than 0 and less than or equal to the number of elements in the array, or refer to a numeric variable with no decimal places.
  - The *Array index cannot itself be a subfield of a multiple-instance array.
- The context specified for *Result or *Factor 2, and the related *Array, must be one of the following contexts:

  ARR

  Valid for both *Result and *Factor 2.

  PAR

  Valid for *Result if the specified field exists as an Output, Both or Neither parameter field on a multiple-instance array parameter. Valid for *Factor 2 if the specified field exists as an Input, Both or Neither parameter field on a multiple-instance array parameter.

  PR*n*

  n is an integer 1–9.

  Same validity as PAR, but used where the function has duplicate parameters.

If *ALL is specified for either *Result or *Factor 2, the following restrictions apply:

- Certain context-specific restrictions apply when you specify here *ALL is specified:
  - If you specify *ALL for *Result, also specify *Factor 2 and vice-versa.
  - If you specify *ALL for *Result, you cannot specify CND and CON as the context for *Factor 2.
  - If you specify *ALL for *Factor 2, you cannot specify NLL as the context for *Result.
- Code is only generated to move a field if all the following conditions apply:
  - The field exists in both the *Factor 2 context and the *Result context
  - If the *Factor 2 context is a parameter context, the field must have a usage of O, B or N
  - If the *Result context is a parameter context, the field must have a usage of I, B or N
- Any target fields (or target array subfields) that exist in the *Result context, but do not exist in the *Factor 2 context, are not changed.

Note: The same field type validation rules apply to *MOVE ARRAY as to *MOVE, in terms of moving numeric fields to non-numeric fields.

From the main Action Diagram Editor screen, you can use the subfile option I=M to insert and prompt the *MOVE ARRAY built-in function.

Considerations for Date and Time Field Types

The following table summarizes conversions the *MOVE and *MOVE ARRAY built-in functions handle automatically for fields that represent dates and times.

To From	NBR	DTE	D8# (DT8) *	DT#	TME	TM#	TS#
NBR	+	+	+	1	+	3	7
DTE	+	+	0	0	–	–	5
D8# (DT8) *	+	2	+	0	–	–	5
DT#	2	0	0	+	–	–	5
TME	+	–	–	–	+	3	6
TM#	4	–	–	–	4	+	6
TS#	8	9	9	9	10	10	+

To From	NBR	DTE	D8# (DT8) *	DT#	TME	TM#	TS#

* Conversions for the shipped D8# and the user-defined DT8 (8-digit internal representation) data types are identical.

Explanations of codes used in this table:

\+ No conversion.

– Does not apply.

0 Type conversion between internal formats.

1 Convert from CYYMMDD.

2 Convert to CYYMMDD

3 Insert delimiters.

4 Remove delimiters.

5 Convert date-to-date part of timestamp, time part is set to 0.

6 Convert time-to-time part of timestamp, date part is not affected.

7 Move numeric value as 6-digit nanoseconds part of timestamp. (Timestamp format is yyyy-mm-dd-hh.mm.ss.nnnnnn.)

8 Move 6-digit nanoseconds part of timestamp to numeric field. (Timestamp format is yyyy-mm-dd-hh.mm.ss.nnnnnn.)

9 Move date part of timestamp to date field.

10 Move time part of timestamp to time field.

- Since no conversion is provided, you can move numeric date fields to and from a NBR field to save or set up internal representations of DTE (CYYMMDD) and 8#/DT8 (YYYYMMDD) fields. This is useful, for interfacing with a 3GL file. The following moves are valid:

Valid Moves

```
DTE   NBR   DTE

D8#   NBR   D8#
```

Note: It is not valid to use a NBR field as an intermediary between DTE and D8# (DT8) since these require a conversion. The following section contains the list of invalid moves.

Invalid Moves

The following moves are not valid:

```
DTE   NBR   D8#

D8#   NBR   DTE
```

When you move a constant into a date, time, or timestamp field, the required format for the constant depends on the type of the target field. The required formats are shown in the following table.

Required Format for Constant	Target Date/Time Field
CYYMMDD or YYMMDD	DTE
YYYYMMDD	D8# (DT8)
YYYY-MM-DD	DT#
HHMMSS	TME
HH.MM.SS	TM#
YYYY-MM-DD-HH.MM.SS	TS#

For more information on the date and time field types, see Defining a Data Model in the chapter "Understanding Your Data Model," Using Fields topic.

Move All

The *MOVE ALL built-in function specifies that all of the fields from one context are to be moved to another context by name. You can specify up to four source contexts.

Note: You cannot use CON, CND, and WRK contexts as *Result contexts for this built-in function.

There are potentially five parameters for this function type:

- Up to four input parameters, which are the four source contexts that can be moved.
- One output parameter which is the *Result context field. For each field in the result context, CA 2E examines the source contexts in the order in which you specify them in the *MOVE ALL action, to determine instances of the same field.

Note: Function fields are not included by *MOVE ALL. You must explicitly move function fields to a new context using the *MOVE built-in function.

The *MOVE ALL built-in function performs a series of moves from one context to another, mapping fields by name.

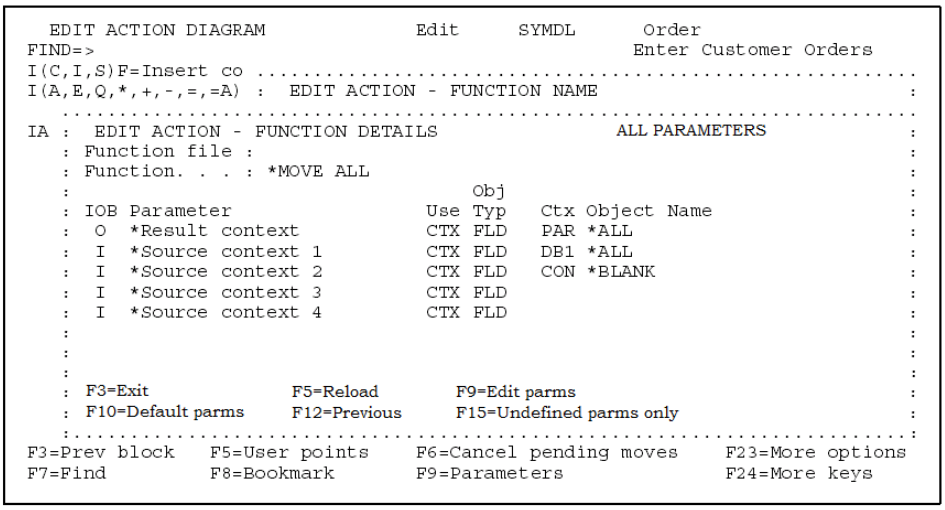
CA 2E implements the *MOVE ALL built-in function as an RPG Z-ADD statement for numeric fields and as a set of MOVEL statements for alphanumeric fields. In COBOL, CA 2E implements the *MOVE function as a set of MOVE statements.

Example

In this example, for a given function, the PAR and DB1 contexts contain slightly different groups of fields as listed in the following table:

PAR. Customer code	DB1. Customer code
PAR. Customer name	DB1. Start date
PAR. Credit limit	DB1. Customer name
PAR. Start date	
PAR. Customer group	

All of the fields in the PAR context could be initialized by a single statement. This is done by inserting a *MOVE ALL function with the following parameters on the Edit Action - Function Details window:



In other words:

```
.--
: PAR = DB1, CON By name                    <<<
'--
```

This is equivalent to:

```
> USER: Process DBF record
:--
:PAR. Customer code = DB1.Customer code     <<<
:PAR. Customer name = DB1.Customer name     <<<
:PAR. Start date = DB1.Start date           <<<
:PAR.Credit limit = CON.*ZERO               <<<
:PAR. Customer group = CON.*BLANK           <<<
'--
```

Multiply

The *MULT built-in function specifies an arithmetic multiplication of two fields.

The *Rounded field allows you to specify whether the result of the multiplication is to be half-adjusted. Specify the condition *ROUNDED for rounding; specify the condition * for no rounding.

Rounding consists of adding 5 (-5 for a negative result) one position to the right of the last decimal position specified for the length of the result field. As a result, rounding occurs only when the number of decimal positions in the result value exceeds the number of decimal positions in the result field length.

There are four parameters for this function type:

- Three input parameters which are the two fields that are to be multiplied *FACTOR1 and *FACTOR2, and the *ROUNDED field.
- One output parameter the *Result field.

FACTOR1, FACTOR2, and the *Result field must all be numeric.

CA 2E implements the *MULT built-in function as an RPG MULT statement. In COBOL, CA 2E implements the *MULT function as a MULTIPLY statement.

Example

This is an example of the *MULT built-in function.

```
> USER: Process detail record
.--
: RCD.Line value = RCD.Quantity * RCD.Price     <<<
'--
```

Quit

The *QUIT built-in function specifies an exit from an action diagram sequence construct or user point; when you specify the *QUIT function, all subsequent steps in the construct (or subroutine) are bypassed.

When you specify the *QUIT function within a sequential construct, CA 2E defines a branch to the end of the subroutine.

If you use the *QUIT function outside of a sequential construct, CA 2E defines a branch to the closest, most recently nested, subroutine which, in many instances, is the user point. To limit the action of *QUIT, you can enclose actions within a sequential construct.

There are no parameters for this built-in function.

CA 2E implements the *QUIT built-in function as a GOTO statement for both RPG and COBOL.

Example

In the following example, the step Update database is not executed if errors occur:

```
.--
: ..Validate fields
: ..Validate relations
: .-CASE
: |-If errors
: | <--QUIT
: '-ENDCASE
: ..Update database
'--
```

Retrieve Condition

The *RTVCND built-in function specifies that the name of a given condition is to be retrieved into a function field. This can be of particular use if you want to show the full description of a condition next to the condition on a panel or report.

The convert condition values command **(**YCVTCNDVAL) creates a file that stores the condition data. This file is in the generation library whenever you execute it.

There are two parameters for this function type:

- One input parameter, which is the status field name
- One output parameter, which is the work field into which the condition is retrieved

Both parameter fields are of usage type vary (VRY).

New Topic

To specify that the description of the current value of the Gender field should be shown in the Gender Name field of an EDTRCD function, you could add a Gender Name field to the function's device design, and insert the following action in the function's action diagram:

```
> USER:
.-
: DTL.Gender name = Condition name of DTL.Gender  <<<
'-
```

You can provide Retrieve Condition functionality with F4 prompting by setting the CUA Prompt (YCUAPMT) model value to *CALC and inserting the *RTVCND built-in function at a CALC: user point in the action diagram.

For more information on the *CALC value for the YCUAPMT model value, see the CA 2E Command Reference, the YCHGMDLVAL command.

Rollback

The *ROLLBACK built-in function allows you to add your own rollback points to a program that is executing under i OS commitment control. Commitment control is a method of grouping database file operations that allows the processing of a database change to be either fully processed (COMMIT) or fully removed (ROLLBACK).

There are no parameters for this function type.

CA 2E implements the *ROLLBACK built-in function as an RPG ROLBK statement and as a COBOL ROLLBACK statement.

For more information on the commit built-in function, see the example with the information on COMMIT in the start of this topic.

Retrieve Field Information

The *RTVFLDINF built-in function specifies that the *meta-information* about a field is to be retrieved into one or more fields.

Meta-information about a field consists of information about the field itself (the field textual name, the DDS name of the field, the field length, and so forth.), irrespective of the current value of the field. This can be of use if you want to build SQL statements to retrieve information from the file containing the field or if you want to write your own utilities to retrieve information about model objects (for instance, for documentation).

The parameters for the *RTVFLDINF function are:

- There is one input parameter (*Field name). This is the 25-character name of the field for which you want to retrieve meta-information. You can specify any valid field context for this parameter, except LCL. Screen contexts (for example, DTL) are allowed, as is the DB1 context.
- Output parameters as follows:
  - *Field name
  - *Field DDS name (see **Note 1**)
  - *Field internal DDS name (see **Note 2**)
  - *Field text
  - *Field surrogate
  - *Field domain surrogate (see **Note 3**)
  - *Field attribute code (see **Note 4**)
  - *Field external data type
  - *Field external length
  - *Field external integers
  - *Field external decimal positions
  - *Field internal length
  - *Field internal integers
  - *Field internal decimal positions
  - *Field contextual name (see **Note 5**)
  - *Field internal data type
  - If any output parameter is specified using the NLL context, no code is generated for them.

Note 1: If the field passed in the input parameter specified with the DB1 context, this returns the name of the field on the file, including the 2-character file prefix. If the field is passed in any other context, this returns the name of the field with the appropriate 2-character program prefix (WU for a field in the WRK context, and so forth)

Note 2: This is the (typically 4-character) name of the field as it appears in the model, without any prefix.

Note: 3: If the field passed in the input parameter is a REF field, this parameter returns the surrogate number of the field to which it is referenced, otherwise it returns the same value as is returned in the *Field surrogate parameter.

Note 4: This is the 3-character field attribute, for example, VNM, TXT, CDE, NBR and so on. If the field is a REF field, the attribute of the referenced field is used.

Note 5: This is the name of the field as it is used in the program. For RPG programs, if the field is specified in the DB1 context, this value is the same as the *Field DDS name, except that it has the rename prefix applied to it instead of the file prefix. For all other contexts, this parameter has the same value as the *Field DDS name. For COBOL programs, this field contains the fully qualified name of the field as it is used in the program.

Set Cursor

The *SET CURSOR built-in function allows you to explicitly position the cursor on any field on the device design by specifying the field name and the context to which it belongs. In addition, the *SET CURSOR built-in function allows you to control cursor positioning based on the occurrence of errors.

There are two input parameters for this function type:

- Field name on which the cursor is positioned
- *Override error field that is conditional and can take a value of *YES or *NO. It determines whether cursor positioning takes place based on the occurrence of an error

For more information on device designs, see the chapter "Device Designs."

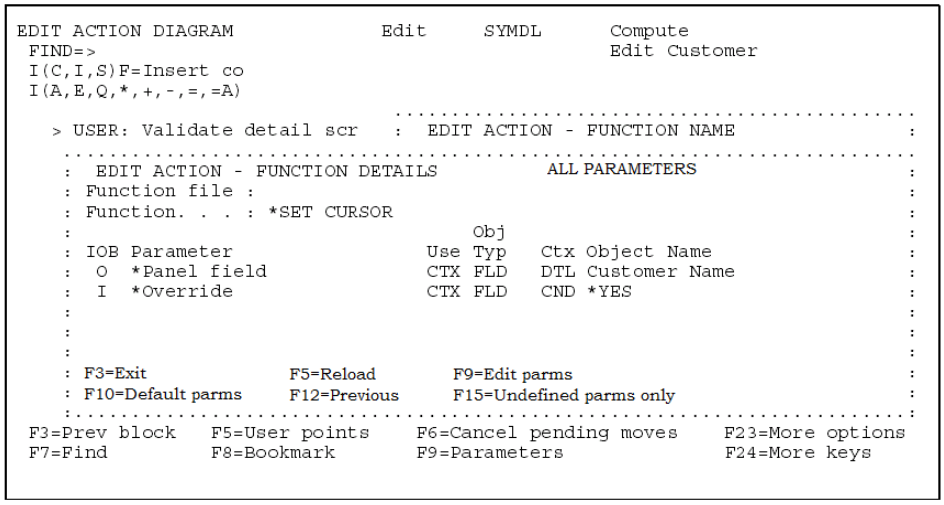
Note: You cannot use *SET CURSOR to override cursor positioning due to errors generated by display files such as values list errors on status fields. These errors are implemented below the level of the application program in the display device or file.

The following is an example of the Set Cursor function.

In this example, if the Customer Status field is Active, move the cursor to the Customer Name field.

```
EDIT ACTION DIAGRAM             Edit     SYMDL      Driver
 FIND=>                                                Prompt Driver
 I(C,I,S)F=Insert construct           I(X,O)F=Insert alternate case
 I(A,E,Q,*,+,-,=,=A)F=Insert action   IMF=Insert message
      > USER: Validate relations
      .--                                                                <<<
      . -CASE                                                            <<<
      . -DTL.Customer status is Active                                   <<<
      . -Set Cursor: DTL.Customer Name (*Overrride=*YES)                 <<<
      . -ENDCASE                                                         <<<
      '_




F3=Prev block   F5=User points    F6=Cancel pending moves    F23=More options
F7=Find         F8=Bookmark       F9=Parameters
```

The following diagram shows the parameters for the example.


EDIT ACTION DIAGRAM
Edit
SYMDL
Compute
FIND=>
Edit Customer
I(C,I,S)F=Insert co
I(A,E,Q,*,+,-,=,=A)
> USER: Validate detail scr
EDIT ACTION - FUNCTION NAME
EDIT ACTION - FUNCTION DETAILS
ALL PARAMETERS
Function file :
Function. . . : *SET CURSOR
Obj
IOB Parameter
Use Typ
Ctx Object Name
O *Panel field
CTX FLD
DTL Customer Name
I *Override
CTX FLD
CND *YES
F3=Exit
F5=Reload
F9=Edit parms
F10=Default parms
F12=Previous
F15=Undefined parms only
F3=Prev block
F5=User points
F6=Cancel pending moves
F23=More options
F7=Find
F8=Bookmark
F9=Parameters
F24=More keys

Substring

The *SUBSTRING built-in function allows you to extract a portion of a character string from one field and insert it into another.

There are five parameters for this function type:

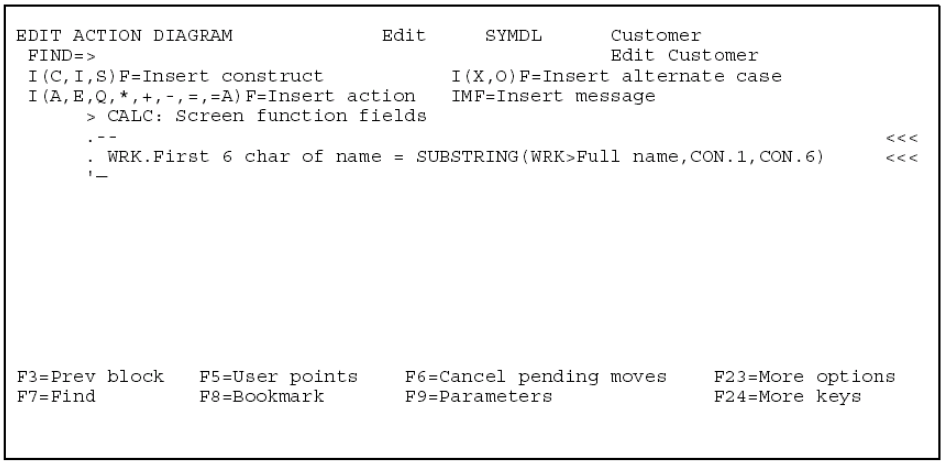
- Three input parameters which are a character string, *String 1, denoting the character substring that is to be extracted, the *From position field denoting the position from which the extraction of the character string occurs, and the *For length field denoting the length of the character substring to be extracted.
- Two output parameters which are the *Resulting string into which the extracted substring is inserted and a *Return code to determine the result of the attempted insertion. Return code is an implied parameter and is set to condition *NORMAL when insertion completes successfully.

The *For Length field has two special conditions: *FLDLEN which indicates that the entire length of the field or constant is to be used and *END which means that the extraction occurs from the position of the *From position field to the end of the field or constant.

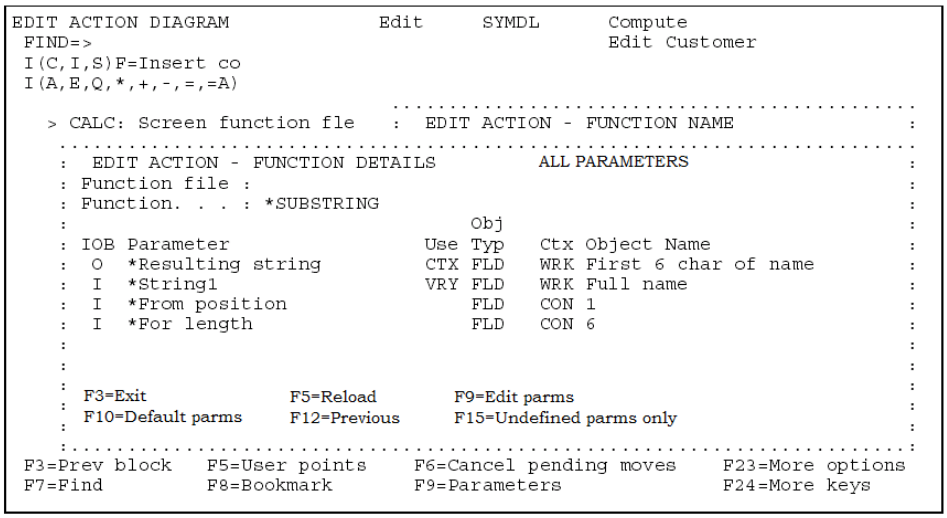
CA 2E implements the *SUBSTRING built-in function in RPG using the SUBSTR statement in COBOL using the STRING statement. For COBOL 85, it uses a reference modification and for COBOL, 74 it is uses string manipulation.

The following is an example of a substring function.

This example extracts the first six characters from the Full Name field.



The following diagram contains the parameters for the previous example.



Subtract

The *SUB built-in function specifies an arithmetic subtraction on two operands, *FACTOR1 and *FACTOR2.

There are three parameters for this function type:

- Two input parameters which are the two operands, *FACTOR1 and *FACTOR2.
- One output parameter which is the *Result field containing the result of the subtraction.

All three parameters must be numeric field types.

CA 2E implements the *SUB built-in function as an RPG SUB statement and as a COBOL SUBTRACT statement.

Time Details

The *TIME DETAILS built-in function returns information about a given time. You specify the type of information you need using the *Time Detail parameter.

There are three parameters for this function type:

- Two input parameters
  - *Time is the time for which information is to be returned. If it is of type NBR, it is interpreted as the elapsed time since 0 am.
  - *Time detail type determines the meaning of the output parameter, *Time detail.
- One output parameter, *Time detail, returns the requested information for the specified time.

The possible values for the *Time detail type parameter and the effect of each on the meaning of the result are summarized in the following table.

*Time Detail Type Values	Effect on the *Time Detail Parameter
*SECONDS	An integer from 0 to 59 specifying the number of seconds in the specified time (*Time).
*ELAPSED SECONDS	An integer containing the number of elapsed seconds since 0 am for the given time (*Time).
*MINUTES	An integer from 0 to 59 specifying the number of minutes in the specified time (*Time).
*ELAPSED MINUTES	An integer specifying the number of elapsed minutes since 0 am for the specified time (*Time).
*HOURS	An integer from 0 to 23 specifying the number of hours in the specified time (*Time).
*HHMM	The number of hours and minutes represented by the given time (*Time) in HHMM format.
*HHMMSS	The number of hours, minutes, and seconds represented by the given time (*Time) in HHMMSS format.
*PM?	An integer that indicates whether the given time is am or pm. The possible values are 1=pm and 0=am.

Time Increment

The *TIME INCREMENT built-in function lets you add a quantity to a given time. It is the converse of the *ELAPSED TIME built-in function.

The *TIME INCREMENT built-in function performs the operation:

```
*Time1 = *Time2 + *Elapsed Time
```

There are four parameters for this function type:

- Three input parameters
  - *Time2 specifies the beginning time. If it is of type NBR, it is interpreted as the elapsed time since 0 am.
  - *Elapsed time specifies the quantity to be added to *Time2.
  - *Time unit specifies the meaning of the *Elapsed time input parameter. Refer to the table at the end of this description.
- One output parameter, *Time1, specifies the ending time. If it is of type NBR, it is interpreted as the elapsed time since 0 am.

  The number of hours in the sum is factored by 24 to produce an integer from 0 to 23. In other words, if the number of hours is 24 or greater, the hours are divided by 24. The final number of hours in the *Time1 parameter is the remainder of the division. For example, if the sum is 64 hours and 32 minutes, the result in *Time1 is 16 hours and 32 minutes (64/24=2 + a remainder of 16).

The possible values for the *Time unit parameter and the effect of each on the meaning of *Elapsed time are shown in the following table.

*Time Unit Values	Effect on the *Elapsed Time Parameter
*SECONDS	An integer specifying the number of seconds to add to the specified time (*Time2).
*MINUTES	An integer specifying the number of minutes to add to the specified time (*Time2).
*HOURS	An integer specifying the number of hours to add to the specified time (*Time2).
*HHMM	The number of hours and minutes, in HHMM format, to add to the specified time (*Time2).
*HHMMSS	The number of hours, minutes, and seconds, in HHMMSS format, to add to the specified time (*Time2

Calculation Assumptions and Examples for Date Built-In Functions

Since months and years do not contain equal numbers of days, date calculations involving these units are adjusted to conform to common sense standards rather than to pure mathematical accuracy. This section presents assumptions made to the output results for the *DATE INCREMENT and *DURATION built-in functions. A set of examples explain possibly confusing results and show recommended function usage.

CA 2E makes the following assumptions to provide consistent results for the *DATE INCREMENT and *DURATION functions independent of the input value:

- Duration between "today" and "today" equals 0 days.
- Duration between "today" and "tomorrow" equals 1 day.
- Since mathematical necessity requires that either the Start date or the End date not be counted when calculating date duration or date increment, CA 2E does not count the Start date (*Date2).

  Note that this is significant only when you have explicitly excluded specified dates from calculations using selection parameters or a date list.

Business and Everyday Calendars

Since the calculation units (months/years) are not always equally long, the idea of a business calendar and an everyday calendar are introduced here to help explain the results produced by the *DATE INCREMENT and *DURATION built-in functions.

Business Calendar

If you specify *DAYS for the *Duration Type parameter, CA 2E bases its calculations on a user-defined "business" calendar. You define a business calendar using the *Excluded Days of Week and *Date List Name parameters to specify working days, non-working days, holidays, and other special days for your business.

The resulting "day-centric" calculations are always mathematically accurate because all units (days) are equally long. In other words, date calculations based on a business calendar are easily understood.

Everyday Calendar

If you specify a value other than *DAYS for the *Duration Type parameter, CA 2E bases its calculations on an "everyday" calendar in which all seven days of each week are included in the calculations. Date Lists and *INCLUDED/*EXCLUDED selections are not used by definition; so all issues associated with these parameters can be ignored.

The resulting "month-centric" calculations are not always mathematically accurate because the units involved are not equally long. The results of such calculations are often approximate because CA 2E adjusts them to common sense standards rather than to mathematical accuracy. For example, December 31, 1995 (*Date2) incremented by 2 months (*Duration) returns a result of February 29, 1996 (*Date1) rather than the arithmetically correct February 31, 1996!

The remainder of this section gives examples of specific assumptions CA 2E makes to adjust results to common sense standards and to produce consistent results.

*DATE INCREMENT Rules and Examples

*DATE INCREMENT performs the following operation:

```
Start date + Increment = End date
```

- If a Start date is incremented by one unit (month/year), the End date is the same day in the next unit.

Start date	Increment	End date
January 05, 1996	1 (*MONTH)	February 05, 1996
January 05, 1996	1 (*YEAR)	January 05, 1997

- If a Start date that is the last day of a month is incremented by one unit (month/year) and the next unit (month/year) is shorter than the current one, the End date is adjusted to the last day of the next unit.

Start date	Increment	End date
March 31 1996	1 (*MONTH)	April 30, 1996
February 29, 1996	1 (*YEAR)	February 28, 1997

As a result, the End date for the one-unit increment case is always within the contiguous unit, which can be the next or previous unit depending on the sign of the increment.

- The one-unit (month/year) increment case can be generically expanded to any combination of month(s) and/or year(s)

Start date	Increment	End date
May 31 1996	4 (*MONTH)	September 30, 1996
December 30, 1993	102 (*YEAR)	February 28, 1995

- When the Start date is the last day of the unit (month/year) it is always associated with the last rather than with the same day of the End unit.

Start date	Increment	End date
February 29 1996	–1 (*MONTH)	September 30, 1996
December 30, 1993	102 (*YEAR)	February 28, 1995

Start date	Increment	End date
February 29, 1996	–1 (*MONTH)	January 31, 1996 (not January 29)
February 29, 1996	2 (*MONTH)	April 30, 1996 (not April 29)

- The *DATE INCREMENT result can be reversed for any day other than the last day of the month by simply changing the sign of the increment. Following is an example where the operation is reversible.

Start date	Increment	End date
January 13, 1996	1 (*MONTH)	February 13, 1996
February 13, 1996	–1 (*MONTH)	January 13, 1996

Following is an example where the operation is not reversible.

Start date	Increment	End date
Jnuary 29, 1996	1 (*MONTH)	February 29, 1996
February 29, 1996	–1 (*MONTH)	January 31, 1996

- The most confusing effect deriving from processing the last day of a month is that different Start dates (usually close to the last day of the Start month) give the same End date (always the last day of the End month) when incremented by the same unit.

Start date	Increment	End date
January 29, 1996	1 (*MONTH)	February 29, 1996
January 30, 1996	1 (*MONTH)	February 29, 1996
January 31, 1996	1 (*MONTH)	February 29, 1996

This is a good example of the approximate calendar calculations mentioned previously.

Note: The everyday calendar can be widely used for business purposes. For example, to process a bank statement when the billing cycle is defined as a time interval between the first and the last day of every month.

- When the increment unit is composed of days, months, and years (*YYMMDD), then calculation is broken into two steps. First, the Start date is incremented by the months/years. Second, the adjusted intermediate date is incremented by days.

Start date	Increment	End date
November 29, 1993	10315 (*YYMMDD)	March 15, 1995

Following are the steps used to produce this result; namely, the Start date is incremented subsequently by 1 year, then by 3 months, and then by 15 days:

	Start date	Increment	End date
1.	November 29, 1993	1 (*YEARS)	November 29, 1994
2.	November 29, 1994	3 (*MONTHS)	February 28, 1995 (Note that the last day is adjusted.)
3.	February 28, 1995	15 (*YYMMDD)	March 15, 1995

Note: The (*YYMMDD)' increment may not always be equal to (*DAYS); for example, if days were excluded from the calendar using a Date List or a selection parameter. However, the (*YEARS) and the (*MONTHS) always equal (*YYMMDD) and the (*YYMMDD).

*DURATION Rules and Examples

*DURATION performs the following operation:

```
End date - Start date = Duration
```

The *DURATION function result is often not as obvious and easily predictable as the *DATE INCREMENT result. For example, what is the duration expressed in *MONTHS between Start date (December 31, 1995) and End date (February 29, 1996)? The function returns two months even though three months are involved in the calculation. This is another example of the Everyday calendar's approximation.

The remainder of this section gives examples of specific assumptions CA 2E makes to adjust results to common sense standards and to produce consistent results.

- The *DURATION function counts a month if the number of covered days is greater than or equal to the number of days in the End date month. A Duration month is defined as the number of days in the End date month. Note that the first day of a duration interval is never counted.

Start date	End date	Duration	Actual Days
December 31, 1995	January 31, 1996	1 (*MONTHS)	31
December 19, 1995	January 23, 1996	1 (*MONTHS)	35
December 28, 1995	January 23, 1996	0 (*MONTHS)	26
January 31, 1996	February 29, 1996	1 (*MONTHS)	29

- The *DURATION function counts a year if any twelve consecutive months are covered. This is the definition of a Duration year. Note that the first day of a duration interval is never counted.

Start date	End date	Duration	Actual Days
December 31, 1995	December 31, 1996	1 (*YEARS)	365
December 31, 1996	December 31, 1997	1 (*YEARS)	365
June 30, 1996	June 30, 1997	1 (*YEARS)	365

- For the *YYMMDD duration type, DD represents the number of days not included in either the Duration year (YY) nor the Duration month (MM).

Start date	End date	Duration	Actual Days
November 12, 1995	March 23, 1997	1 (*YEARS)	497
November 12, 1995	March 23, 1997	16 (*MONTHS)	497
November 12, 1995	March 23, 1997	104 (*YYMM)	497
November 12, 1995	March 23, 1997	10410 (*YYMMDD)	497

Following are the steps used to produce this result.

a. Calculate years. From within the range of dates, identify any whole years consisting of 12 consecutive whole months. For example, December 1, 1995 to November 30, 1996 = 1 year.

b. Calculate months. From within the remaining dates, identify any whole months; in this example, December 1996 + January 1997 + February 1997 = 3 months.

c. Calculate days. All that remains from the original date range are parts of the first and last months in the range; namely, November 12, 1995 to November 30, 1995 (18 days) and March 1, 1997 to March 23, 1997 (23 days). The total number of remainder days is 41 (18 + 23 = 41). Since 41 days is greater than the Duration month of 31, add one month to the total number of months (3 + 1 = 4 months).

   Recall that the DD portion of the *YYMMDD duration type represents the number of days not included in either the Duration year (YY) or the Duration month (MM). In other words, Remainder days (DD) – Duration month = 10 days (41 – 31 = 10).

The final result in *YYMMDD format is 10410.

- The *DURATION function calculates one month if both the Start and End dates represent the same day in contiguous months.

Start date	End date	Duration	Actual Days
December 1, 1995	January 1, 1996	100	31
January 1, 1996	December 1, 1995	–100	–31
February 12, 1995	March 12, 1995	100	28
February 12, 1996	March 12, 1996	100	29

Application of this rule sometimes causes different, but close, Start and End dates to return the same duration.

Start date	End date	Duration	Actual Days
May 19, 1996	June 20, 1996	102	32
May 20, 1996	June 20, 1996	100 (2)	31
May 21, 1996	June 20, 1996	100	30
May 22, 1996	June 20, 1996	029	29

(1) The End month, June, has 30 days; so the Duration month in this example is 30. DD=Actual Days–Duration month=02.
(2) The arithmetic result 101 is adjusted to 100 since the Start and End dates represent the same day in contiguous months.

- The *DURATION function calculates one year if both Start and End dates represent the same day and month in contiguous years.

Start date	End date	Duration	Actual Days
December 8, 1995	December 8, 1996	10000	366
February 1, 1995	February 1, 1996	10000	365
February 1, 1996	February 1, 1995	–10000	–365

Application of this rule sometimes causes different, but close, Start and End dates to return the same duration.

Start date	End date	Duration	Actual Days
February 28, 1995	February 29, 1996	10000	366
February 28, 1995	February 28, 1996	10000 (1)	365

(1) This result was adjusted since the Start and End dates represent the same day and month in contiguous years.

Understanding Contexts

A CA 2E context is a grouping of the fields that are available for use at a particular processing step in an action diagram. A context specifies which instance of a particular field is to be used. Fields can be referenced for use as parameters in functions and in conditions to control functions.

Different contexts are available at different points of the action diagram depending on the function type, the stage of processing, and the particular nature of the user point; that is, whether a subfile control format or a record detail format is being processed.

Each type of context is identified by a three-letter CA 2E code. For example, PAR for parameter fields, CON for constant fields, and WRK for work fields. The following pages describe the context types and their usage.

Database Contexts

DB1

The Database One (DB1) context contains the fields that are in the first, or only, format of the access path to which a function is attached.

Any field in the access path format is available for processing in the DB1 context.

The DB1 context is available to all function types that perform update or read functions on a database file after reading and before writing to the database file. Those functions are CRTOBJ, CHGOBJ, DLTOBJ, and RTVOBJ.

In the generated source code, the generated names of fields from the DB1 context are prefixed by the format prefix of the appropriate DBF file.

For example, if the following relations are present in an access path for the Company file:

```
FIL   Company  REF   Known by   FLD  Company code   CDE
FIL   Company  REF   Has        FLD  Company name   TXT
```

Company name is present in the DB1 context of functions using that access path and can be used in action diagrams (where access to the database is allowed), for example:

```
WRK.  Company name = DB1.  Company name                <<<
```

DB2

The Database Two (DB2) context contains the fields that are in the second format of the access path to which a function is attached.

Any field in the second access path format is available for processing in the DB2 context.

The DB2 context is available only to functions that are attached to a Span (SPN) access path; the DB2 context applies to the detail file (or second format) in the SPN access path. The DB2 context is available only within an EDTTRN function to access the secondary format.

In the generated source code, the generated names of fields from the DB2 context are prefixed by the format of the appropriate DBF file.

Consider the following example. A Span (SPN) access path is created for Order and Order Detail and the second (detail) format for Order Detail contains the following relations:

```
FIL   Order detail   CPT  Owned by   FIL   Order           QTY
FIL   Order detail   CPT  Refers to  FIL   Product         REF
FIL   Order detail   CPT  Has        FLD   Order quantity  QTY
```

Order Quantity is present in the DB2 context of functions using that access path. It may be used in the action diagram where access to the database is allowed, for example:

```
WRK. Order quantity = DB2. Order quantity        <<<
```

ELM

The ELM context contains the fields defined for the last-accessed element of a specified array. This context is valid only in the *CVTVAR built-in function and may be specified for either the input or output parameter.

Since a single element of an array is equivalent to a data structure, you can use the ELM context

- To decompose a field into a structure. A move of an element of an array to a field constitutes a move of the array's structure to the field.
- To group a set of fields into a single field.

Note: You must define a key for an array even if the array holds a single element.

Following is more information on these two usages.

Move from a Field to a Structure

- The array is the output of the *CVTVAR function.
- To ensure that the array index is not corrupted by the move, the array must be defined as a single-element array.
- The *CVTVAR function is implemented as a MOVEL operation from the field to the data structure (array). Blanks are moved to the array element before the data is moved.

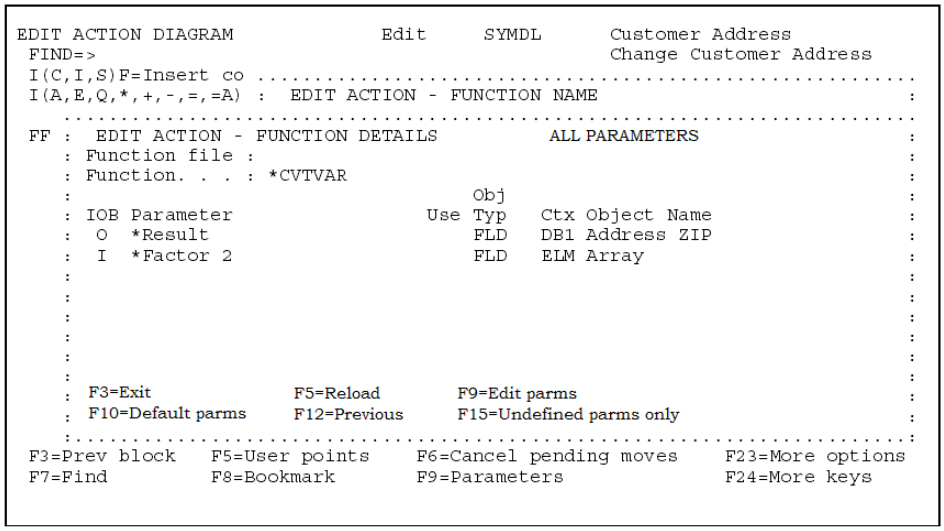
Here is an example of how this operation might look in an action diagram.

```
EDIT ACTION DIAGRAM             Edit      SYMDL      Vendor
 FIND=>                                                 Edit Vendor
 I(C,I,S)F=Insert co ..........................................................
 I(A,E,Q,*,+,-,=,=A) :  EDIT ACTION - FUNCTION NAME                           :
    ..........................................................................
 FF :  EDIT ACTION - FUNCTION DETAILS          ALL PARAMETERS                 :
    : Function file :                                                         :
    : Function. . . : *CVTVAR                                                 :
    :                                    Obj                                  :
    : IOB Parameter                 Use Typ    Ctx Object Name                :
    :  O  *Result                       ARR    ELM Arrray                     :
    :  I  *Factor 2                     FLD    DB1 Address ZIP                :
    :                                                                         :
    :                                                                         :
    :                                                                         :
    :                                                                         :
    :                                                                         :
    :                                                                         :
    : F3=Exit              F5=Reload        F9=Edit parms                     :
    : F10=Default parms    F12=Previous     F15=Undefined parms only          :
    :.........................................................................:
 F3=Prev block   F5=User points    F6=Cancel pending moves    F23=More options
 F7=Find         F8=Bookmark       F9=Parameters              F24=More keys
```

Move from a Structure to a Field

- The array is the input of the *CVTVAR function.
- The *CVTVAR function moves the last accessed array element of the named array to the named field. In this case, the array may contain multiple elements.
- The *CVTVAR function is implemented as a MOVEL operation from the associated data structure (array) to the field.

Here is an example of how this operation might look in an action diagram.



For more information:

- On the *CVTVAR built-in function, see section, Understanding Built-In Functions.
- On decomposing and recomposing character data, see this chapter, Understanding Built-In Functions topic, *CONCAT and *SUBSTRING functions.
- On arrays, see the section Using Arrays as Parameters (see page 284), and see the chapter "Defining Arrays" in the *Building Access Paths Guide.*

Device Contexts

KEY

The Key (KEY) context contains the fields that are on the key panel display of the device functions that have key panels. These keys apply to the record functions: Edit Record (EDTRCD 1,2,3) and Display Record (DSPRCD 1,2,3).

All the key fields on the access path to which the function is attached are available in this context, along with any associated virtual fields. If you map parameters to the device panel (as mapped or restrictor parameters), they are also available in this context.

The shipped field *CMD key is present in this context only in the exit program user point. It is also present in the detail format. You cannot add function fields to a key panel.

Consider the following example. An Edit Record function is defined for an Employee file, using an access path that has keys defined as follows:

```
FIL  Company   REF  Known by   FIL   Company code   CDE
FIL  Company   REF  Has        FLD   Company name   TXT

FIL  Employee  REF  Owned by   FIL   Company        REF
FIL  Employee  REF  Known by   FLD   Employee code  CDE
FIL  Employee  REF  Has        FLD   Employee name  TXT
```

Company name is a virtual field on the Employee file through the Owned by relation. The fields available in the KEY context of the action diagram for the function are:

```
*CMD key
Company code
Company name
Employee code
```

DTL

The Detail (DTL) context contains fields on the display panels of device functions that have single, non-subfile detail panels such as EDTRCD, or multiple, non-subfile panels such as EDTRCD2.

All of the fields from the access path to which the function is attached are available in the DTL context. If you map parameters to the device panel as mapped or restrictor parameters, they are also available in this context.

The shipped field, *CMD key, is present in this context. If you add any function fields to the detail panel display, these fields are available in the DTL context.

The DTL context is available in the action diagrams of PMTRCD, EDTRCD, and DSPRCD. This context is only available after the key has been successfully validated.

Consider the following example. An Edit Record function is defined on a Stock Item file using an access path, which includes the following relations:

```
FIL  Stock item  REF  Known by  FLD  Stock item code  CDE
FIL  Stock item  REF  Has       FLD  Stock item qty   QTY
FIL  Stock item  REF  Has       FLD  Item price       PRC
```

Stock Item, Qty field and Item Price Field are present in the DTL context for that function and could be used in the action diagram; for instance:

```
WRK. Stock value = DTL. Stock item * DTL. Item price     <<<
```

2ND

The Second Detail panel (2ND) context contains the fields that are on the second detail panel display of a device function that has a multi-part panel design attached to it, such as EDTRCD2 and DSPRCD2.

All of the fields from the access path to which the function is attached are available in the 2ND context. If you map parameters to the device panel as mapped or restrictor parameters, they also are available in this context.

The 2ND context is only available in the action diagrams of the following function types:

- Edit Record 2 panels
- Edit Record 3 panels
- Display Record 2 panels
- Display Record 3 panels

3RD

The Third Detail panel (3RD) context contains the fields that are on the third detail panel display of a device function that has a multi-part panel design attached to it, such as EDTRCD3 and DSPRCD3.

All of the fields from the access path to which the function is attached are available in the 3RD context. If you map parameters to the device panel as mapped or restrictor parameters, they also are available in this context.

The 3RD context is only available in the action diagrams of the following function types:

- Edit Record 3 panels
- Display Record 3 panels

CTL

The Subfile Control (CTL) context contains the fields that are in the subfile control record of the device functions that have a subfile panel display such as Display File or Edit Transaction.

The fields available in the CTL context depend on the function type, the access path used by the function, and whether you have specified restrictor parameters for the function. The *CMD key shipped field and any parameters specified as mapped or restrictor parameters are present on the CTL context.

If the function is attached to a SPN access path, the CTL context contains all of the fields from the header format on the access path.

If the access path is a RTV or a RSQ and the function is:

- Display type (Display File or Select Record), all of the fields on the access path are available in the CTL context unless any of those fields have been dropped from the panel subfile control format. The key fields can be used as positioner parameters, the non-key fields can be used as selectors.
- Edit type (Edit File), all of the key fields on the access path are available in the CTL context unless any of those fields have been dropped from the panel subfile control format. These fields can be used as positioner parameters. For key fields, which are restrictor parameters for the function, any associated virtual fields are also present on the CTL context.

If you defined any key fields as restrictor parameters, any virtual fields are also available in the CTL context.

The CTL context is available in the action diagrams of all functions that have a subfile panel display:

- Display File (DSPFIL)
- Display Transaction (DSPTRN)
- Edit File (EDTFIL)
- Edit Transactions (EDTTRN)
- Select Record (SELRCD)

The CA 2E shipped field *CMD key provides the means of specifying that a specific piece of logic is to be executed whenever the user presses a particular function key. The *CMD key is a status (STS) field and already has defined conditions for many possible function or control key combinations.

For more information on a list of the default function keys, see this module, in the chapter, "Modifying Device Designs."

Consider the following example of the use of the *CMD key.

To specify the use of a function key to call another program, you should insert the relevant processing at the appropriate point in the action diagram, for example:

```
> USER: Process command keys
.--

:  .-CASE                                    <<<
:  |-CTL. *CMD key is CF06                   <<<
:  | Print detailed report                   <<<
:  '-ENDCASE

'--
```

Consider the following example of the indirect use of a *CMD key.

If you want to be able to remap the function keys of your application to another standard, you should use LST conditions in place of direct function key conditions. For example, you could:

1. Define a LST condition called Display Print.
2. Condition your action diagram using this condition.

   Assign a function key such as F6 to the Display Print condition.

```
> USER: Process command keys
.--

:  .-CASE                                    <<<
:  |-CTL. *CMD key is Display print          <<<
:  | Print detailed report                   <<<
:  '-ENDCASE

'--
```

RCD

The Subfile Record (RCD) context contains the fields that are in the subfile record of device functions that have a subfile panel display such as Display File or Edit Transaction.

The fields available in the RCD context depend on the function type and the access path used by the function. The *SFLSEL shipped field is present on the RCD context unless you specifically remove it from the function options. Mapped parameters specified as mapped or restrictor parameters are present on the RCD context.

If you have a SPN access path, the RCD context contains all of the fields from the detail format on the based-on access path.

If you have a RTV or RSQ access path, all of the fields from the based-on access path are available in the RCD context.

The RCD context is available in the action diagrams of all functions that have a subfile panel display:

- Display File (DSPFIL)
- Display Transaction (DSPTRN)
- Edit File (EDTFIL)
- Edit Transaction (EDTTRN)
- Select Record (SELRCD)

The shipped *SFLSEL (subfile selector) field provides a selection column field for subfiles. You can optionally remove this field from the panel design using the Edit Function Options panel. The *SFLSEL field is a status (STS) field and is shipped with some predefined conditions.

For more information on a list of SFLSEL conditions, see this module, in the chapter, "Modifying Device Designs."

These conditions can be displayed and changed by pressing F9 at the Edit Field panel of the *SFLSEL field. If you always see the *SFLSEL conditions using list conditions, you are able to reuse a single selection code for multiple purposes.

CUR

The Current Report Format (CUR) context contains all of the fields that are in a report format of a Print File or Print Object function.

The report functions have header and total formats for each key level in the based-on access path, except for the lowest level key, as well as a final total format. For example, an access path with three key fields, would have three total formats:

Headings for level 1 (1HD)
Headings for level 2 (2HD)
Detail record, level 3 (RCD)
Totals for level 2 (3TL)
Totals for level 1 (4TL)
Final totals (ZTL)

The detail record format contains all of the fields from the based-on access path while the heading and total formats for each level contain the key field for that level and any associated virtual fields. These fields are available in the action diagram where the processing for a particular format occurs in the CUR context. In addition, any function fields that you have added to the report format are available in the CUR context.

The CUR context differs from the NXT context in that the NXT context contains fields that are present in the report format representing the next level break in the report. The CUR context specifies fields that are being processed at a given point in the action diagram while the NXT context specifies fields being processed in the following format.

The CUR context is only available in the action diagrams of the following function types:

- Print File (PRTFIL)
- Print Object (PRTOBJ)

For example, in a report to print out an Order, if Order Qty and Product Price are fields present on the access path to which the Print File function is attached and Order Line Val is a function field attached to the Detail Record (RCD) format, the following processing might be inserted in the action diagram at the point where detail records are processed:

```
> USER: Process detail record
.--
: CUR.Order line val = CUR.Order qty * CUR. Product price
'--
```

NXT

The Next Report Format (NXT) context defines a context relative to the CUR context for report functions. The NXT context contains fields that are in the next active report format (not dropped format) that is one level break lower. You could use the NXT context to specify the placement of the result of a function field (SUM, MIN, MAX, or CNT) on the appropriate report total format. You can only use the NXT context for result fields.

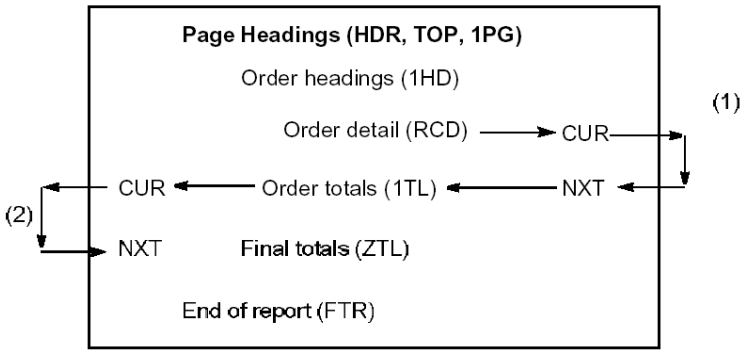
The NXT context is only available in the action diagrams of the following function types:

- Print File (PRTFIL)
- Print Object (PRTOBJ)

For example, in a report to print out an Order: if the Order detail format contains the field, Order Line Value, and Total Order Value is a SUM function field based on this field, the same function field can be attached to two different formats:

- The Order detail format to sum into the Order totals format (for the Order no. key level).
- The Order total format to sum into the Final totals format.

In both cases, the instance of the input field is from the CUR context while the result field is placed on the NXT context.



The function field is inserted into the action diagram of the report function at the points where each format is processed.

```
> USER: Process Order detail
.--                                                        <<<
: Total order value

:     O   Total order value  NXT   Total order value
:     I   Order line value   CUR   Order line value

'--
```

```
> USER: Process Order no. totals
.--                                                        <<<
: Total order value

:     O   Total order value  NXT   Total order value
:     I   Order line value   CUR   Order line value

'--
```

The parameters for each call to the function field can be accessed using the Edit Action - Function Details window. For example, Order Line Value is input from the CUR context and Total Order Value is output to the NXT context:

EDIT ACTION - FUNCTION DETAILS ALL PARAMETERS

Function file . . : *FIELD
Function . . . . : *Total order value

IOB	Parameter	Use	Obj Typ	Ctx	Object Name
O	Total order value		FLD	NXT	Total order value
I	Order line value		FLD	CUR	Order line value

F3=Exit F5=Reload F9=Edit parms
F10=Default parms F12=Previous F15=Undefined parms only

If a report format is dropped from a CA 2E report design (using the Edit Design Formats panel), the NXT context retains its meaning. Any fields in the context are automatically assumed by CA 2E to be on the next highest format level. Thus, in the previous example: if the Order Totals format was dropped, the Order Line Value function field would be assumed to total directly onto the Final totals format.

Literal Contexts

CND

The CA 2E Condition (CND) context enables you to specify that a particular field condition value is to be supplied as a field value in one of the following ways:

- As a parameter to a function.
- As the condition that controls a conditional or iterative construct in the action diagram.

All field conditions that are attached to a field are available in the CND context for that particular field.

The CND context is available in the action diagrams of all function types. You can use the CND context for function parameter and action diagram condition fields. However, you cannot use the CND context for result fields; that is, you cannot move another field value into a condition context field.

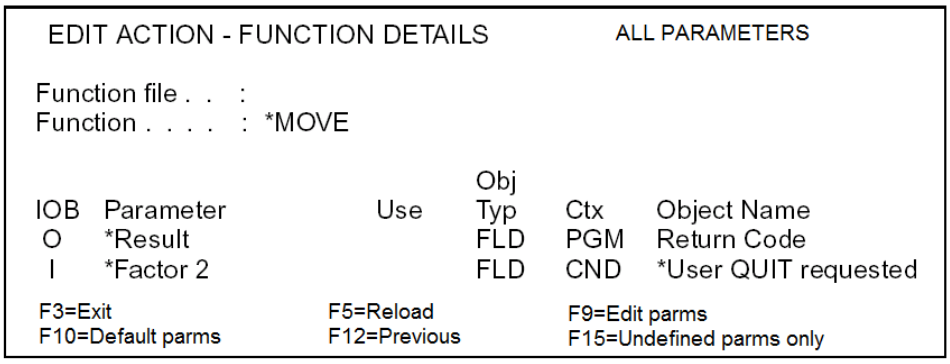
To use a CND context field as a parameter, you specify that the conditions attached to the field be passed as a parameter value to a result field.

To use a CND context field as a condition, you specify processing in a CASE construct if a particular field condition is attached to a field. For example, if CND *Return code is *User QUIT requested, then *Exit program.

The following example shows both uses of the CND context.

As a parameter:

The CA 2E supplied status field *Return code has several conditions attached to it; for example, *User QUIT requested. Any one of the conditions can be used as a parameter on the Edit Action - Function Details window:



This is shown in the action diagram as:

```
PGM. *Return code = CND. *User QUIT request   <<<
```

As a condition:

If *User QUIT requested is the name of a condition attached to the *Return code field, then an example of conditioning a CASE statement might be:

```
.—CASE                                          <<<
: PGM.*Return code is *User QUIT requested      <<<
: Exit program                                  <<<
‘—ENDCASE
```

CON

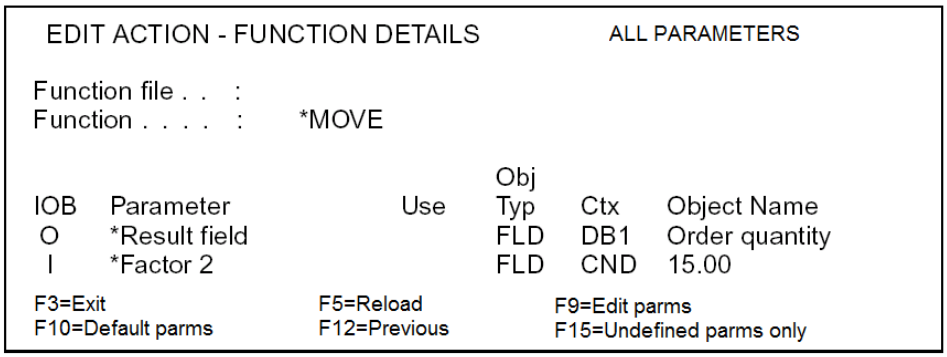
The Constant (CON) context contains any constant or literal values that you want to specify.

You only use CON context values to specify input values to fields. There are also some restrictions associated with the usage of this context:

- You cannot use the CON context with status (STS) fields. You should use the CND context with STS fields.
- Numeric constants must be less than or equal to ten characters in length, including the decimal point and sign. A maximum of five characters is allowed after the decimal point.
- Alphanumeric constants must be less than or equal to twenty characters in length.

The CON context is available in the action diagrams of all function types.

To specify that a numeric field Order Quantity is to be set to a value of 15:



This is shown in the action diagram as:

```
DB1.Order quantity = CON.15.00                    <<<
```

System Contexts

JOB

The Job (JOB) context contains system fields that supply execution time information about the job that executes the HLL program implementing a function. You cannot add additional fields to the JOB context. You would use this context primarily to define system data to a particular field, such as job date, user name, or job execution start time.

You can only use JOB context fields for input to other functions. They cannot be changed.

The fields that appear in the JOB context are provided in a shipped file called *Job Data.

Field	Attr	Role
*USER	VNM	System name of job user
*JOB	VNM	System name of job
*PROGRAM	VNM	System name of HLL program
*Job number	NBR	Job number
*Job submitted/start date	DTE	Date job submitted
*Job exec start date	DTE	Date job started executing
*Job exec start time	TIME	Time job started executing
*Job date	DTE	Time and date job started executing
*Job year	NBR	Current year of job date
*Job month	NBR	Current month of job date
*Job day	NBR	Current day of job date
*Job time	TME	Current time of job date
*Job hour	NBR	Current hour of job date
*Job minute	NBR	Current minute of job date
*Job second	NBR	Current second of job date
*Function main file name	VNM	System name of function’s main file (1)
*Function main file lib	VNM	System name of lib. containing file (1)
*Function main file mbr	VNM	System name of file member (1)
*Function main lib/file	CDE	Function’s main library and library file (1)
*Current	RDB	For DRDA
*Local	RDB	For DRDA

Field	Attr	Role
(1) Not valid for SQL		

The JOB context is available in the action diagrams of all function types.

PGM

The Program context (PGM) contains system fields that control the execution of a function. An example of a PGM field would be *Program Mode which determines the program mode in which a program executes.

The fields that appear in the PGM context are defined in a system file called *Program Data.

The PGM fields are:

Fields	Attr	condition	DSP Value
*Program mode	STS	*ADD	ADD
		*AUTO	AUTO
		*CHANGE	CHANGE
		*DISPLAY	DISPLAY
		*ENTER	ENTER
		*SELECT	SELECT
*Return code	STS	*NORMAL	*BLANK
		*User QUIT requested	Y2U9999
*Record data changed	STS	*NO	N
		*YES	Y
*Record selected	STS	*NO	N
		*YES	Y
*Reload subfile	STS	*NO	N
		*YES	Y
*Scan limit	NBR	-	-
*Defer confirm	STS	*Defer confirm	Y
		*Proceed to confirm	N
*Print format	STS	*Do not print format	N
		*Print format	Y

Fields	Attr	condition	DSP Value
*Continue transaction	STS	*NO *YES	N Y
*Next RDB	VNM	-	-
*cursor filed	VNM	-	-
*cursor row	NBR	-	-
*cursor column	NBR	-	-
*Re-read Subfile Record	STS	*NO *YES	N Y
*Initial call	STS	*NO *YES	N Y
*Sbmjob override string	TXT	–	–
*Sbmjob job name	VNM	–	–
*Sbmjob job user	VNM	–	–
*Sbmjob job number	CDE	–	–
*Synon work field (15,0)	NBR	-	-
*Synon work field (15,2)	NBR	-	-
*Synon work field (15,5)	NBR	-	-
*Synon work field (17,5)	NBR	-	-
*Synon work field (17,7)	NBR	-	-
*Synon work field (17,9)	NBR	-	-

The Display value can be translated to other national languages if appropriate.

You can use the fields in the PGM context to control processing within function.

Each field is discussed briefly following:

*Program Mode

The *Program Mode field specifies the current mode of a program. This field can be used to provide an override to the default initial mode of CA 2E functions and to condition processing according to the current mode.

For example, when you first enter an Edit File or Edit Record function, the program is in *CHANGE Mode unless there are no records in the file to be edited. In this case the program is in *ADD Mode. If you want the end user to be in *ADD Mode regardless of the presence of records in the file, you can override the default in the User Initialization part of the action diagram for the Edit function using the built-in *MOVE function such as:

```
PGM. *Program mode = CND.*ADD      <<<
```

*Return Code

The *Return Code field contains a return code that is passed between all standard functions. This field may be used to communicate with the calling and called functions.

You can add extra conditions to the *Return Code field. A special facility is provided on the Edit Field Conditions panel for this field, which allows you to define conditions by selecting from existing message functions. The conditions created have the same name as the selected message function (for example, user quit requested) and have the message identifier used to implement the message (USR0156) as a condition value.

If you want to specify a fast exit (for example, pressing F15 when using a function that is called by another function exits you from both functions), the *Return Code field would be used in the PGM context as follows:

1. In the called program you would use the *Exit Program built-in function to specify an exit from the program when F15 is pressed. The parameter for this function is the *Return code field which you would specify as the condition *User QUIT requested:

```
.—CASE                                                    <<<
|—CTL.*CMD key is CF15                                    <<<
|  Add new records function                               <<<
|  *Exit program — return code CND. *User QUIT requested <<<
‘—ENDCASE
```

1. In the calling program you would specify, immediately after the call to the subsidiary function, that if the return code returned corresponds to the condition *User QUIT requested, quit the (calling) program:

Execute subsidiary function

```
.—CASE                                                    <<<
|—PGM. *Return code is *User QUIT requested               <<<
| *Exit program — return code CND. *User QUIT requested <<<
‘—ENDCASE
```

*Reload Subfile

The *Reload Subfile field specifies that a subfile is to be reloaded before redisplay. CA 2E standard functions normally only reload subfiles if it is required by the default processing. You can use this field if you want to force a subfile reload.

For example, if a Display File function calls a subsidiary function that adds records to the database, you may want the subfile to be reloaded on return so that the new records are included in the display. To force a subfile reload, you should move the condition *YES to the *Reload subfile field immediately after the call to the subsidiary function; this causes the subfile to be reloaded on return to the Display File function.

```
.—CASE                                          <<<
|—CTL.*CMD key is CF09                          <<<
| Add new records function                      <<<
| *PGM. *Reload Subfile = CND. *Yes             <<<
‘—ENDCASE
```

*Record Data Changed

The *Record Data Changed PGM context field specifies whether the data for the current record has changed. The value *Yes means the record data has changed; the value *No means that the record data has not changed; it is initialized to ‘ ‘ (blank). The database record is updated only when the value of this field is *Yes.

Note: Checking for unchanged record data is done only if the Null Update Suppression function option is Y or A.

You can access the *Record Data Changed field from all functions that contain an embedded CHGOBJ function, such as EDTFIL and EDTTRN. It is valid only in the Data Read and Data Update user points in the action diagram of CHGOBJ functions; other uses give invalid results. Two ways to use this field are:

- You can test its value to conditionally perform actions that depend on whether data has been changed as shown in the example following.
- You can manually set this field to conditionally force or suppress a database record update.

The following example illustrates a common technique for setting an audit stamp. The field, Timestamp, is set to the current date and time only if other fields within the record have been changed. In other words, if the value of the *Record Data Changed field is *Yes, then the audit stamp is written to the file.

The Timestamp field has the following characteristics:

- It is of type TS#.
- It is on the update access path for the CHGOBJ function.
- It is a Neither parameter on the CHGOBJ function, which means it is not passed into the routine and must be set by the action diagram code within CHGOBJ.

```
> USER: Processing before Data update
.—                                                          <<<
:.> Only set the Timestamp field if other data has changed <<<
:.—CASE                                                     <<<
:|—PGM.*Record Data Changed is *Yes                         <<<
:| DB1.Timestamp=JOB.*Job date                              <<<
:| DB1.Timestamp=JOB.*Job time                              <<<
:’—ENDCASE                                                  <<<
‘_
```

For more information on the CHGOBJ function, see this module, in the chapter "Defining Functions," CHGOBJ—Database Function topic.

*Record Selected

The *Record Selected field specifies that a record read from the database is to be processed. In CA 2E standard functions that read multiple records from the database, for instance to load a subfile, you can add user-defined processing to specify which records are to be included. The record selected field allows you to indicate whether a record is to be included or omitted.

For example, if you want to add your own selection criteria to the loading of the subfile in a Display File or Edit File function, you should insert it into the USER: Initialize Subfile Record from DBF Record part of the Load Next Subfile Page routine in the action diagram for the function. For instance, if you want to specify that records with a zero date field are to be omitted:

```
> Load next subfile page

       REPEAT WHILE
     -Subfile page not full
     PGM. *Record selected = CND. *YES
     Move DBF record fields to subfile record
     > User: Initialize a subfile record from DBF record
      .
      .—
      : .-CASE
      . | -DB1.Date of birth is *ZERO
      : | PGM. *Record Selected = CND. *NO
      : .-ENDCASE
      ‘—
        .-CASE
        | PGM. *Record Selected is YES
        | Write subfile record
        ‘-ENDCASE
      Read next DBF record
      ENDWHILE
```

*Scan Limit

The *Scan Limit field specifies a limit to the number of records that are to be read at a time. If additional selection is applied when reading records from the database (for instance, on the previous *Record selected field), then a limit can be specified on the number of records that can be unsuccessfully read.

The default value for the Scan Limit is 500. For example, to specify that the Scan Limit is to be 100, use the built-in function *MOVE to set the *Scan Limit field to this value in the USER: Initialization exit point of an action diagram:

```
  PGM. *Scan limit = CON. 100                    <<<
```

*Defer Confirm

The *Defer Confirm field only applies to functions where a confirm prompt is available. In such functions there are likely to be occasions when you want to suppress the confirm prompt and subsequent processing. On these occasions, *Defer Confirm causes the panel to be redisplayed as if the user had replied No to the confirm prompt. The effect is the same, even if the confirm prompt option is not specified on the Edit Function Options panel.

For example, in an Edit File function, if a line is selected with a Z, you would probably not want to display the confirm prompt on returning to the Edit file display. To prevent the program from displaying the confirm prompt, move the condition *Defer Confirm to the *Defer Confirm field. This causes the confirm and update part of the processing to be skipped.

```
.—CASE                                          <<<

|—RCD.*SFLSEL is *Zoom                          <<<
| Display details function                      <<<
| *PGM. *Defer confirm = CND. *Defer confirm    <<<
‘—ENDCASE                                       <<<
```

*Print Format

The Print Format field specifies whether a format from a report is to be printed. The *Print Format option only applies to Print File and Print Object functions. There may be instances when you want to select more records from the database file for processing by the function (controlled with the *Record Selected field) than you want to be printed.

For example, if you want a Print File function to print either a detailed or a summary report, depending on the value of an input parameter to the function, you can control which formats are printed in the two reports by means of the *Print Format field.

```
> USER: On print of detail format               <<<

.--                                             <<<
: .—CASE                                        <<<
: | -PAR.Report type is Summary                  <<<
: | -*PGM.*Print format = CND.*Do not print format  <<<
: ‘—ENDCASE                                     <<<
‘--                                             <<<
```

*Continue Transaction

The *Continue Transaction field is applicable for transaction functions such as EDTTRN and DSPTRN that have input restricted key fields. This field can be used to perform the equivalent of reload subfile for these two functions without returning to the key panel. This can be done within the action diagram of the function by inserting the following code in the *EXIT PROGRAM user point:

This piece of action diagram logic enables the subfile of an EDTTRN or DSPTRN to be reloaded and the subfile redisplayed without having to return to the key panel.

```
> USER: Exit program processing                    <<<

.—                                                 <<<
: .—CASE                                           <<<
: |-CTL.*CMD key is *Exit                          <<<
: |-OTHERWISE                                      <<<
: | PGM.*Continue transaction = CND.*No            <<<
: | <--QUIT                                        <<<
: ‘—ENDCASE                                        <<<
‘_
```

*Next RDB

The *Next RDB field is applicable for functions with distributed functionality. If the value of *Next RDB is not blank, the value is used to establish a connection prior to performing database access.

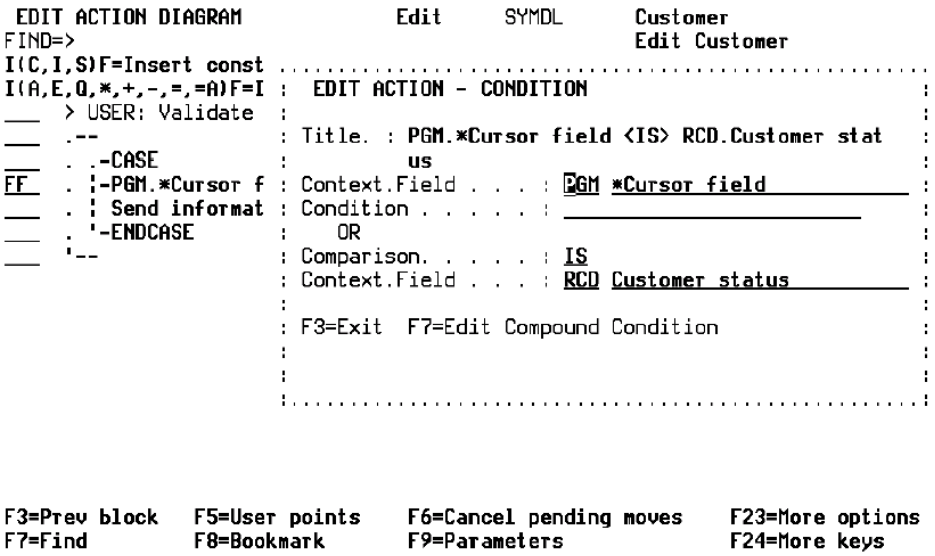
For more information on using DRDA, see Generating and Implementing in the chapter "Distributed Relational Database Architecture."

*Cursor Field

The *Cursor Field always contains the name (DDS name, for example Z1ABCD) of the field where the cursor is currently positioned.

IS Comparison Operator	You can use the IS comparison operator to determine whether the cursor is positioned on a specific field on the panel by comparing the PGM
Cursor Field to the field. You can use the IS comparison operator in any condition statement.	

The following is an example of the IS comparison operator. This example shows A test that determines if the cursor is positioned on the Customer Status field.



Use the PGM *Cursor Field to test for cursor position on subfile fields with the RCD context. In this case, both the field and the subfile record must match the cursor position for the test to be true. This is implemented automatically by CA 2E.

When testing for cursor position on subfile records, the logical place to insert the CASE statement in the action diagram would be

USER: Process subfile record (or equivalent user point.)

Some additional logic is required to test for a cursor position in the DSPFIL, EDTFIL, and SELRCD function types. This is discussed under the example for PGM *Re-read subfile record field.

PGM *Cursor Field is available for reference only. For example, you are not able to move conditions or values into this field.

*Cursor Row and *Cursor Column

The *Cursor Row and *Cursor Column field always contains the relative numeric row and column that the cursor is currently on. As with *Cursor Field, these fields are available as reference fields only.

*Re-Read Subfile Record

The *Re-read Subfile Record field can be used to force the reprocessing of subfile records whether or not they have been changed. This is particularly useful when testing for cursor position on a subfile within the DSPFIL, EDTFIL, and SELRCD function types.

Differences in Subfile Processing Between EDTTRN and DSPTRNs Compared to DSPFIL, EDTFIL, and SELRCDs

The EDTTRN and DSPTRN functions load all records in a subfile before the panel appears. They also re-process all records in the subfile when the user requests validation of the data.

In the example described under PGM *Cursor field, this means that the CASE statement is tested for every subfile record and is able to determine the exact subfile record and field on which the cursor is currently positioned.

The DSPFIL, EDTFIL, and SELRCD functions load records one page at a time.

They only re-process records that have been modified or touched by the end user. This processing enables these functions to perform efficiently and ensures that records that have not been modified are not processed. This means that the USER: Process Subfile Record (or equivalent user point) is examined for EDTTRN and DSPTRN functions for every record in the subfile.

Since DSPFIL, EDTFIL, and SELRCD functions only process changed records, the CASE statement is only tested for those records that have been flagged as modified.

In order for the cursor position to be detected on subfile records in these function types, you must change the subfile record or you must flag the subfile record to be re-read in any event.

To achieve the latter, another field PGM *Re-read subfile record can be used.

To ensure that subfile records are re-processed, the PGM *Re-read Subfile Record field should be set to *Yes as follows:

```
 PGM.*Re-read Subfile Record = CND.*YES

 <--QUIT
```

If the DSPFIL had no post-confirm pass, the following additional action diagram logic would be required at the end of User: Process Subfile Record (pre-confirm) user point. This would ensure that the subfile records are re-processed again if the subfile was re-loaded after the processing pass of the records.

You can condition the setting of *Reread Subfile record based on a set of conditions. This could be used to pre-select records that meet certain status criteria.

For example:

```
  > USER: Process subfile record (Pre-confirm)
```

```
.--
:.—CASE
:|-RCD.Order value is GT CTL.Customer Max Order Value
:| PGM.*Re-read subfile record = CND.*No
:| Send information message — Order & 1 will not be accepted
:|-*OTHERWISE
:| WRK.Highlight field = CND.*YES Order Value
:'—ENDCASE
'--
```

*Synon Work field (15,0) to (17,9)

These fields are available as numeric fields and have the lengths as specified in their description. These fields can be used as computational work fields.

*Synon Work field (17,7) is the default value intermediate result field that is used when a compute expression is initially created.

For more information, see "Editing Compute Expressions."

*Initial Call

The *Initial Call field allows you to detect whether the function is being invoked for the first or subsequent times.

This field is only of use if the function option Closedown Program has been set to *No in which case the value of PGM *Initial Call is *Yes. For subsequent calls to this function the value is *No.

COBOL programs also use this field. Nested COBOL programs remain under the control of the calling program. The *Initial Call field is set to *YES the first time a nested COBOL program is called. Subsequent calls set the field to *NO.

In order to execute logic that is only performed once when a function is first invoked, the following action diagram logic can be inserted:

```
>USER: Initialize program


.—CASE
|-PGM.*Initial Call is *Yes
| Load Product Info array — Product
'—ENDCASE
```

The field PGM *Initial Call is available for reference only and is set to *No by the function automatically.

For more information and examples on how to use the PGM *Initial Call field, see Building Access Paths in the chapter "Defining Arrays."

*Sbmjob override string

The *Sbmjob field provides dynamic overrides of SBMJOB parameters when a job is submitted for batch processing from within an action diagram. Only EXCEXTFUN, EXCUSRPGM, and PRTFIL functions can be submitted for batch processing from an action diagram.

Note: This feature does not support function calls that contain multiple-instance array parameters.

For more information on submitting jobs from within an action diagram, see Submitting Jobs Within an Action Diagram (see page 568).

*Sbmjob job name, *Sbmjob job user, *Sbmjob job number

These PGM context fields facilitate additional processing for jobs submitted from an action diagram; for example, handling spool files, follow-on updates, lock manipulation, and any other processing that requires submitted job information.

Function Contexts

PAR

The Parameter (PAR) context contains the fields that you define as parameters for the current function. This includes the function whose action diagram you are currently editing.

You can specify function parameters using the Edit Function Parameters panel. When you define a field as a function parameter, CA 2E automatically adds the field to the PAR context of the function, but availability of the fields associated with the PAR context is user-point dependent.

If you define parameters for a particular function, then the PAR context is available at all points in the action diagram of that function.

Consider the following example. If you create a Display file function on a Horse file, you could specify Horse code as an input parameter and Race date and Race time as output parameters:

```
 EDIT FUNCTION PARAMETER DETAILS      My Model
Function name. . : Display Racing results    Type : Display file
Received by file : Race Entry                Acpth: Races for a Horse
Parameter (file) : Race Entry                Passed as: KEY
   ?  Field                        Usage   Role   Flag error
   _  Horse code                     I      RST
   _  Race date                      O      MAP
   _  Race time                      O      MAP

SEL: Usage: I-Input, O-Output, B-Both, N-Neither, D-Drop.
     Role: R-Restrict, M-Map, V-Vary length, P-Position. Error: E-Flag Error.
F3=Exit
```

The Horse code could then be used as an input field and the Race date and Race time as output fields at appropriate places in an action diagram. For example:

```
CTL.Horse code = PAR.Horse code                    <<<



PAR.Race date = DTL.Race date                      <<<
PAR.Race time = DTL.Race time                      <<<
```

LCL

The LCL context provides the ability to define variables that are local to a given function. It is available in all situations where the WRK context is available and all fields defined in the model are presented for selection.

Although neither parameter provide a method of defining local work variables, this method requires additional effort and is not as flexible.

The LCL context provides an alternative to the WRK context and avoids two major pitfalls of the latter. Since the WRK context is global

- A WRK field can inadvertently be changed by an internal function
- It encourages the dangerous practice of using WRK fields to communicate among functions without using parameters

Set the Parameter Default Context (YDFTCTX) model value to *LCL to use LCL rather than WRK as the default context for parameter defaulting in the action diagram editor.

The model-level default context is displayed in the Subfile Control area when an action has undefined parameters. This field can then be changed prior to pressing F10. Available default contexts are:

- LCL – All parameters use the LCL context.
- WRK – All parameters use the WRK context.
- NLL – Output parameters use the NLL context.

Both and Input parameters use the LCL context.

Special Considerations

- Internally, the generators create a new field for each function in which a LCL field is declared. As a result, although the LCL context defines fields that are local to a particular function, another function can change the value of the field. For example, a LCL field passed to another function as a Both parameter can be changed by the called function.
- A given program can have up to 9999 LCL parameters.
- For internal functions that are not implemented as shared subroutines, only one LCL variable is generated; in other words, all instances of the internal function share the same local variable. Thus, the LCL context is only logically local to a particular function.

NLL

The NLL context is available in all situations as a target for output parameters. The generators process this context by allocating a field from a separate sequence of field names. Such fields are local to a particular function in the same way as LCL context fields.

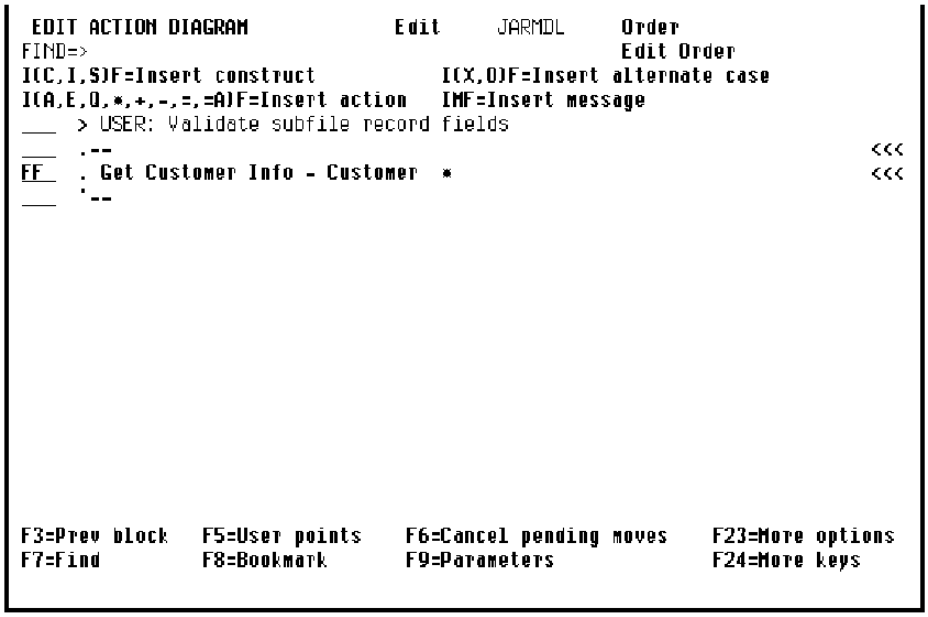
If you change an Output parameter that is supplied using the NLL context to Input or Both, the action diagram editor displays a message when the function parameters are prompted.

Benefits

- You can safely discard output parameters without worrying about whether they are overwriting a work or local field used elsewhere.
- You can trim your model of unnecessary functions; for example, a suite of RTVOBJ functions, each of which returns a different output, can be replaced with a single general purpose RTVOBJ.
- You do not need to define special fields to use as discard targets.
- Since the NLL context is output-only, it can be used repeatedly to receive output from multiple functions.

Generic RTVOBJ

The NLL context encourages use of a single general purpose RTVOBJ rather than a suite of RTVOBJ functions, each of which returns a different output.



When this function is called from within the action diagram of another function, set the output parameters you want returned to an appropriate context and set all others to the NLL context.

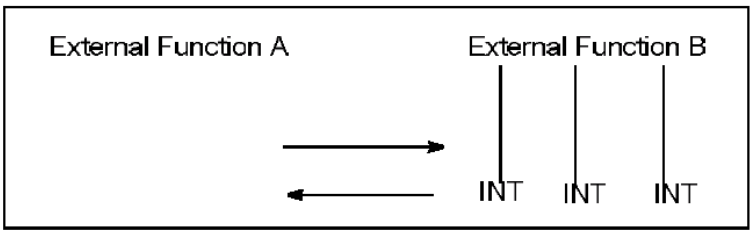
```
 EDIT ACTION DIAGRAM              Edit      JARMDL      Order
FIND=>                                                 Edit Order
I(C,I,S)F=Insert co ..........................................................
I(A,E,Q,*,+,-,=,=A) :  EDIT ACTION - FUNCTION NAME                          :
__ ..........................................................................
IA :  EDIT ACTION - FUNCTION DETAILS   ALL PARAMETERS                       :
__ : Function file : Customer                                               :
   : Function. . . : Get Customer Info                                      :
   :                                                                        :
   :                                      Obj                               :
   : IOB Parameter                  Use Typ   Ctx Object Name               :
   :  O  Customer address           MAP FLD   NLL Customer address          :
   :  O  Customer city              MAP FLD   NLL Customer city             :
   :  O  Customer postal code       MAP FLD   NLL Customer postal code      :
   :  O  Customer phone number      MAP FLD   NLL Customer phone number     :
   :  O  Customer status            MAP FLD   NLL Customer status           :
   :  O  Customer credit limit      MAP FLD   LCL Customer credit limit      :
   :                                                                        :
   : F3=Exit             F5=Reload      F9=Edit parms                      :
   : F10=Default parms   F12=Previous   F15=Undefined parms only           :
   :                                                                        :
F3 :                                                                        :
F7 :........................................................................:
```

WRK

Work (WRK) context fields are useful to contain work variables for interim calculations or for assigning work data or strings in interim processing.

You can use any field in the data dictionary as a work field. You can add other user-defined fields to the WRK context by adding them to the data dictionary using the Define Objects panel.

The WRK context is available at all points in the action diagram of all function types. WRK context variables are global to the external function and so can be changed at any point by any function.



As shown in the previous example of two external functions, if you include internal functions within an external function an umbrella effect allows the internal and the external function to share the same work field. Consequently, any changes to that external function’s work field could cause changes to the internal functions.

This means that any actions or functions within the external function can change any WRK variable without being passed as a parameter. You should only use WRK variables when there are no intervening function calls that could change the values.

Note: The LCL context provides a method of defining local work variables.

For example, you could use a work field to keep a count of the number of records processed:

```
WRK.Counter = WRK.Counter + CON.1          <<<
```

ARR

ARR is similar to the WRK context but is used only for a multiple-instance array parameter. This context is only available on a function call statement in the Action Diagram of functions of type Execute External Function.

By passing a parameter as an array, multiple instances of data can be passed in or out, in a single call to a function. For example, if a customer record structure is defined in the Customer array on the *Arrays file, that array can be used to define a parameter for an Execute External Function (EXCEXTFUN) or Execute User Program (EXCUSRPGM) being passed as RCD (ARRAY). Anywhere from a few to thousands of customer records can be passed in that one parameter in one single function call.

Multiple-instance Arrays and the ARR Context

Arrays are a standard component of the 2E model that you might have utilized in the past. An array is a structure, comprised of multiple fields (array subfields), that has a specified number of elements. Arrays are defined in the *Arrays file.

Earlier, there were three ways to use an array:

- Through the use of CHGOBJ, CRTOBJ, DLTOBJ, and RTVOBJ functions built over an array to process data in the array, treating the array as if it were an access path, where each element of the array equates to a record.
- As a structure that can be used to pass parameters to a function.
- With the *CVTVAR built-in function, you can compose a single string from an array structure of multiple fields. You can also decompose a single string into its constituent fields.

  For more information on these functions, see the *Building Applications Guide* and the *Command Reference Guide.*

With the second and third uses, only a single *instance* of the array structure was referred to – the array was being used only as a structure definition, in other words.

Now we increased the ways you can use arrays:

- Treat an array as a multiple *instance* structure. In other words, as a *true* array with multiple elements within the Action Diagram. In this case use the new *MOVE ARRAY function – individual fields in a variety of contexts can be copied into array subfields in a specified instance of the array and vice-versa.
- Pass a multiple-instance array as a parameter to a function. By passing a parameter in this way, you can pass multiple instances of data in or out as a single parameter, in a single call to a function.

Important! As a CA 2E developer, you need to understand the architectural distinction between the two mechanisms to manipulate array data, despite the ability to use a common structural definition:

- Data can exist and be modified in an array by using database functions (Create Object – CRTOBJ, Delete Object –DLTOBJ, Change Object – CHGOBJ, and Retrieve Object – RTVOBJ) based over the *Arrays file. However, this array data cannot be accessed by the *MOVE ARRAY function.
- Conversely, data can exist and be modified in a multiple-instance array parameter (in the PAR context) and in the ARR context by using the *MOVE ARRAY function. However, that array data cannot be accessed by database functions (Create Object – CRTOBJ, Delete Object –DLTOBJ, Change Object – CHGOBJ, and Retrieve Object – RTVOBJ) based over the *Arrays file.

To make this new functionality possible, we created a new array-related context, ARR context. The ARR context is similar to the WRK context, but is used to define a multiple-instance array. Similar to fields in the WRK context, arrays in the ARR context are initialized during program initialization.The ARR context is available in the following circumstances:

- For use with the *MOVE ARRAY built-in function (see page 472).
- When passing a multiple-instance array to a function that has a multiple-instance array parameter. For more details, see Enhanced Array Support (see page 538).

Enhanced Array Support

Earlier, you could define a parameter to a function to be passed as a FLD, RCD, or KEY.

- **FLD–**Each field specified as a parameter on the parameter details display is passed as an individual parameter.
- **KEY–**A single parameter, where the length is derived from the keys of the specified access path or array, is passed. An externally defined data structure is used to define the parameter.
- **RCD–**A single parameter, where the length is derived from the specified access path or array format, is passed. The parameter contains all the fields which are individually specified as parameters using the parameter details display. An externally defined data structure is used to define the parameter.

You can now pass certain parameters as an array.

By passing a parameter as an array, multiple instances of data can be passed in or out, in a single call to a function. For example, if a customer record structure is defined in the Customer array on the *Arrays file, that array can be used to define a parameter to an Execute External Function (EXCEXTFUN) or Execute User Program (EXCUSRPGM) being passed as RCD (ARRAY), any amount of customer records can be passed in that one parameter, in one single function call.

The Edit Function Parameters Panel and the Edit Function Parameter Details Panel were updated to accommodate this enhancement.

Enhanced Array Support Terms

To assist you with understanding this enhancement, we use two new descriptive terms throughout the CA 2E documentation:

Multiple-instance array parameter

Describes when a parameter is passed as an array (when the "Pass as Array" flag is set to 'Y'). The parameter contains multiple instances of data, where each instance contains all the fields which are individually specified as parameters using the parameter details display.

Single-instance array parameter

Describes when a parameter defined using an array is not passed as an array (when the *Pass as Array* flag is not available or is set to blank). The parameter contains all the fields which are individually specified as parameters using the parameter details display.

Enhanced Array Support Restrictions

This new functionality has a number of fundamental restrictions:

- Only functions of type Execute External Function (EXCEXTFUN) and Execute User Program (EXCUSRPGM) allow parameters to be passed as array.
- Parameters can only be passed as array when the parameter structure is defined using an array based over the *Arrays file.
- Parameters can only be passed as array when they are being passed as RCD or KEY.
- No fields can be dropped on a parameter being passed as an array.
- Does not allow a multiple-instance array parameter in a function call, in both ARR and PAR context, except when calling an EXCEXTFUN or EXCUSRPGM which has multiple-instance array parameter. Additionally, the call must be from the top-level action diagram of an EXCEXTFUN.
- The Submit job (SBMJOB) feature and Y2CALL command do not support function calls that contain multiple-instance array parameters.

When working with two functions, function A and function B, for example, you can model in the action diagram of function A a call to function B, where B has a parameter interface passed as an array. In this case these additional restrictions apply:

- Function A must be of type EXCEXTFUN, and function B must be of type EXCEXTFUN or EXCUSRPGM.
- The parameter context must be PAR or ARR and the array name must exactly match on the parameter definition of A and B.
- If a parameter is passed as an array on A it must be passed as an array on B.
- The parameter must be passed as RCD on both A and B, or KEY on both A and B.
- Though the usages of the subfields on a parameter passed as an array can be mixed, the usages must be compatible, such that the calling function can call the called function.
- In general, the 2E tool prevents the use of modeling scenarios that cannot be successfully generated.

Note: For more details, see the sections for Edit Function Parameters Panel (see page 543) and Edit Function Parameter Details Panel (see page 544), or see the chapter "Defining Function Parameters" in the *Building Applications* guide.

Performance Considerations for Multiple-Instance Array Parameters

When using multiple-instance array parameters (MIAPs), the following performance-related considerations apply:

General performance considerations when using MIAPs

Even though a MIAP can have many instances, only a single pointer is passed by the operating system to the program, as is the case with a non-MIAP parameter. Therefore, in terms purely of the parameter being passed as a *normal* parameter or as a MIAP, there is no additional performance *hit* to using MIAPs.

Performance considerations in programs that use MIAPs

Any *Neither* parameters that are passed to a function are explicitly initialized in the ZZINIT subroutine in that function. In the case of a MIAP parameter containing *Neither* subfields, this initialization occurs for every field, in every element in the MIAP.

Performance considerations in programs that call other programs that use MIAPs

When one program calls another and passes a structure parameter (RCD or KEY), CA 2E generates code in the calling program to initialize the intermediate structures used to pass the parameters to the called program, to load those structures from the variables specified in the Action Diagram and to unload those structures into the return variables. This code is generated whether the parameter is defined as a MIAP or as a *normal* parameter.

When the parameter is defined as a MIAPs

- If all the MIAP subfields are defined with the same usage (I, O, B or N), then the generator loads the entire array structure into and out of the intermediate structure using a single MOVE. By contrast, if the MIAP subfields have different usages, each field must be moved separately. Since the MOVE is repeated for every element of the MIAP, this can affect your performance. Additionally, the amount of code generated for a MIAP with varying-usage subfields can be significantly greater.
- As with fields passed as parameters in non-MIAPs, MIAP subfields with a usage of *Neither* are explicitly initialized prior to the call – this occurs even if all the subfields have a usage of *Neither*.
- As with fields passed as parameters in non-MIAPs, if any of the MIAP subfields are ISO-type fields (DT#, TM# or TS#), code is automatically generated to explicitly initialize them, whether or not they are passed with the same usage as all other subfields within the MIAP.

To ensure the best possible performance when using MIAPs, follow these guidelines as closely as you can

1. All subfields within a MIAP should be defined with the same usage (I, O or B).
2. MIAP subfields with a usage of Neither should be avoided.
3. ISO-type MIAP subfields should be avoided.

Generated Source

Using either multiple-instance array parameters or the related *MOVE ARRAY built-in function results in additional fields and structures (and the code to initialize and process them) to be generated in your source. These fields and structures can significantly increase the size of the source member. Therefore you can control the level of in-line source commenting through the YGENCMT model value, as follows:

***STD**

A single comment line is generated for each multiple-instance array definition, control structure definition and initialization

***ALL**

A comment line is generated for each multiple-instance array subfield definition, control structure subfield definition and initialization

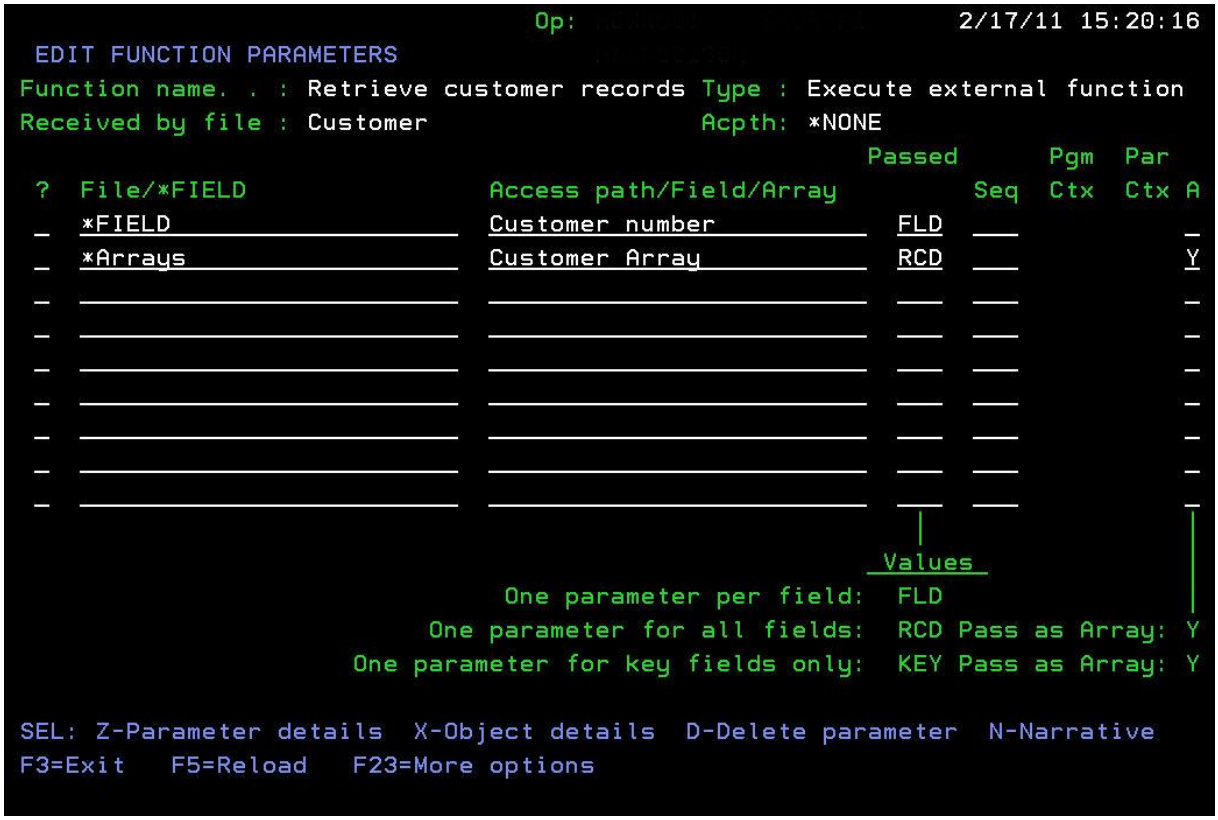
Note: This is the only difference between YGENCMT(*STD) and YGENCMT(*ALL); switching between these two values does not affect any other comment generation within your source code.

Enhanced Array Support Usage

The following sections show how you can use the Enhanced Array Support in CA 2E panels.

Edit Function Parameters Panel

The new functionality allows you to specify, in the Edit Function Parameters panel, that a parameter should be passed as an array, rather than passed as a single-instance structure.



This is done using the new ‘A’ (Pass as Array) field as follows:

- Enter Y when the parameter should be passed as array.
- Leave this field blank when the parameter is a single-instance structure.

The following situations apply when using the 'A' (Pass as Array) field:

- If the function is not an EXCEXTFUN or an EXCUSRPGM, the field is not available on this panel.
- Specifying 'Y' is only valid when the parameter is an array based on the *Arrays file, and only when the parameter is passed as RCD or KEY.

Edit Function Parameter Details Panel

For EXCEXTFUN and EXCUSRPGM only, there are two new values for the *Passed as* field:

- RCD(ARRAY) = Parameter is defined using RCD structure and passed as array.
- KEY(ARRAY) = Parameter is defined using KEY structure and passed as array.

When *Passed as* has a value of RCD (ARRAY) or KEY (ARRAY), the new *Number of elements* field displays the number of elements defined for the array being passed. You can view and modify the array definition by visiting the *Arrays file. If you use option D to drop any fields, an error message displays.

Edit Action Diagram Panel

When a parameter is not passed as an array the behavior of the Edit Action Diagram panel remains the same as previous versions of CA 2E. However, where a parameter is being passed as an array, there is a new single subfile line that indicates an array being passed.

Note: If the called function's parameter interface is modified to toggle the parameter *Passed as Array* field from *Y* to blank, the behavior of the EDIT ACTION – FUNCTION DETAILS changes accordingly to match.

IOB

A, indicating an Array.

Obj Typ

ARR, indicating array.

Ctx

Only PAR or ARR is allowed.

Note: This is an input capable field. ARR is always valid as a choice when the called function's parameter is passed as an array. However, if you select PAR, but the definition of the array on the calling function is incompatible with the definition of the array on the called function, a warning is sent (after PAR is selected) and the change to PAR cannot be saved/completed.

Object Name

Item, indicating the name of the array.

Note: This field is output only.

The array's subfields and their usages are NOT shown on this panel. But you could examine Function B's parameter interface (and detailed usage) via F9=Edit Parms.

This screen shows what the function call will look like for passing an array called *Item*:

When a parameter is being passed as an array,the context defaults to ARR if F10=default parms is used. If the context is changed to PAR, the panel validates that the selected array is available in the PAR context and that all array subfields have compatible usages.

Note: MIAPs do not automatically get defaulted to PAR/PRn context, even when they are available. Your choices for populating MIAP parameters are:

- Manually specify ARR context which is always valid.
- Manually specify PAR/PRn contexts if available.
- Prompt for available contexts.
- Use F10 to default MIAPs to use ARR context.

Understanding Conditions

A condition specifies the circumstances under which an action, a sequential statement, or an iterative statement is to be executed. Conditions define a particular value for a field. The following examples demonstrate how conditions control processing.

A condition controlling a simple action would be an instance where, if a field's condition is met, a simple action takes place.

```
.—CASE

|-RCD.*SFLSEL is *SELECT
| Display record details
‘—
```

A condition controlling a sequence of actions would be an instance within a CASE construct where, if a field's condition is met, a sequence of actions executes.

```
.—CASE

|-RCD.*SFLSEL is *SELECT
|  .--
|  : Display record details
|  ‘--
‘—-ENDCASE
```

A condition controlling an iterative constant would be an instance within a REPEAT WHILE construct where while a field's condition is met, a simple action takes place

```
.= REPEAT WHILE

|-RCD.Status is Held
|  Display record details
‘—-ENDWHILE
```

Similarly, you could define multiple conditions within the same CASE construct to test for various conditions and the actions to take.

```
.—CASE

| -RCD.*SFLSEL is *SELECT
| Display record details
| -RCD.*SFLSEL is *Delete
| Delete record details
| -*OTHERWISE
| Update record details
‘—ENDCASE
```

Condition Types

Condition types allow you to define a particular type of processing based on some form of conditional criteria that you specify in the logic of your action diagram. CA 2E specifies four different condition types that can be used within an action diagram.

Values (VAL) Conditions Type

The Value Conditions (VAL) type is used for conditions that specify a value that a field can receive. You only use the VAL condition type with status (STS) fields.

You specify two related values for a value condition: an internal value that is held in the database file and against which the condition is checked; and an external value that the user enters on the external function application panel. CA 2E generates the necessary code to interpret the values. The internal and external values can have different lengths; you can use the value mapping facility to facilitate translation between the disparate values.

To use value mapping you must specify Y for the Translate field on the Edit Field panel.

For more information:

- On the conditions file and the Convert Condition Value command (YCVTCNDVAL) used to convert the file, see the *CA 2E Command Reference Guide*.
- On VAL, see Defining a Data Model, Using Conditions section in the chapter "Understanding Your Data Model."

Values List (LST) Condition Type

The Values List (LST) Condition type is used for conditions that specify a list of values that a status field can receive. Each condition list consists of one or more value (VAL) conditions.

You can only use the LST condition type for fields of type status (STS). For fields of type STS, CA 2E creates a special list condition *ALL VALUES whenever you define field conditions.

If you specify a value for the Check Conditions prompt on the Edit Field Details panel or the Edit Entry Details panel, CA 2E generates code to ensure that any value that you enter is one of the allowed values.

CA 2E generates, by default, the code necessary to display a list of values for fields of type status (STS) whenever you type **?** in the field or press F4 with the cursor positioned on the field. However, CA 2E only generates the code for the list display if the field is of type STS and if you have defined a check condition for the field.

For more information on LST, see the Using Conditions topic in Defining a Data Model in the chapter "Understanding Your Data Model.".

Compare (CMP) Condition Type

The CA 2E Compare (CMP) condition type is used for conditions that specify a scope of values that a field can receive. The scope of values is defined in terms of a fixed value and an operator. The fixed value is a CA 2E field; the operator is a symbol expressing some form of Boolean logic.

The following is a valid list of operators:

Value	Description
*EQ	equal to
*NE	not equal to
*GT	greater than
*LT	less than
*GE	greater than or equal to
*LE	less than or equal to
*IS	for comparison to PGM *Cursor field

Note: You can use the CMP condition type for field types other than STS

Examples

- Order quantity is GT 10
- Credit limit is LT 1,000.00

For more information on CMP, see Defining a Data Model in the chapter "Understanding Your Data Model," Using Conditions topic.

Range (RNG) Condition Type

The Range (RNG) condition type is used for conditions that specify a range of values that a field can receive. The range of values is defined in terms of two fixed values between which the value must lie including starting and end points. You can use the RNG condition type for field types other than STS.

Example

- Order quantity is between 10 and 100
- Transaction value is GT 25 and LE 250

For more information on RNG, see Defining a Data Model in the chapter "Understanding Your Data Model," Using Conditions topic.

Compound Conditions

CA 2E compound conditions provide you with the ability to use complex condition expressions in any context where a simple action diagram condition is used.

Use Boolean logic operations as AND or OR in condition tests. There are three aspects of compound condition expressions:

1. The ability to AND together or OR together condition tests.

   For example: (a AND b AND c), (a OR b OR c)

2. The ability to parenthesize and mix logical operators.

   For example: (a AND b) OR (c AND d)

3. The ability to test negation.

   For example: (a AND b) OR NOT c

CA 2E provides the following default logical operators for use with compound conditions:

Value	Description
&	AND operator
|	OR operator
(	left parenthesis
)	right parenthesis
!	NOT operator

These operators can be modified by changing the YACTCND model value using the YCHGMDLVAL command.

Defining Compound Conditions

1. Zoom into the user points. At the Edit Action Diagram panel, press F5 to view the user points for the function.

   The Edit User Exit Points window appears.

2. Zoom into a selected user point. Type **Z** next to the selected user point and press Enter.

   The next level of the action diagram appears.

3. Insert a CASE condition. Type **v** at the selected point and press Enter.

   The new case appears.

4. Zoom into the condition. Type **FF** next to the new condition and press Enter.

   The Edit Action - Condition Window appears.

5. Define the compound condition. Press F7.

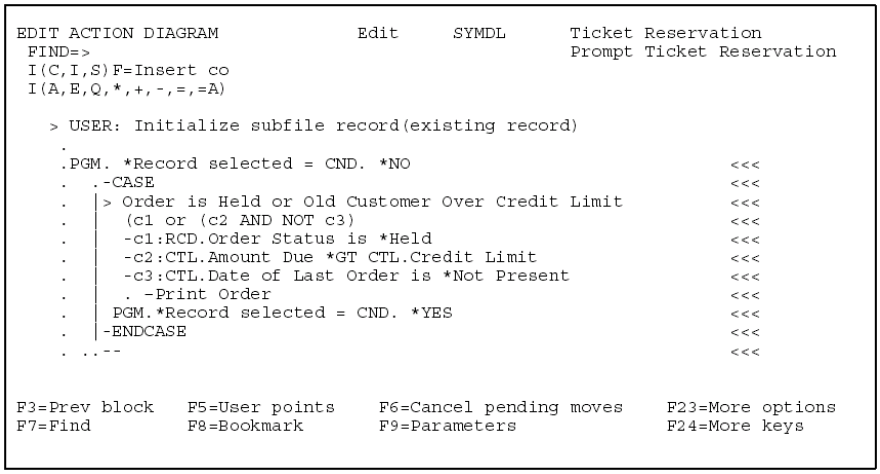
   The Edit Action - Compound Condition panel appears.

6. Enter the compound condition using the logical operators mentioned previously.

   Note: On the Edit Action - Compound Condition panel you have an input-capable, 240-character field.

7. Enter the condition variables on the input-capable line.

   CA 2E creates the undetermined condition statements on the lower portion of the panel. At this point, you can enter **F** against the condition to display the Edit Action - Condition panel at which point you can specify the condition.



Note: Once you define a compound condition in the action diagram, type **F** next to the condition. CA 2E displays the Edit Action - Compound Condition panel.

Understanding Shared Subroutines

The main objective of shared subroutines is to optimize the generation of internal functions that are implemented as subroutines. The first instance of generated source for the function is reused for all subsequent calls to the function within an action diagram instead of being repeatedly regenerated. These changes apply to CHGOBJ, CRTOBJ, DLTOBJ, RTVOBJ, and EXCINTFUN function types.

Some advantages of shared subroutines are:

- The volume of source code is reduced and therefore programs generate faster.
- Fewer subroutines result in smaller and therefore faster executing programs.
- Moving the interface outside shared subroutines facilitates changes required for ILE (Integrated Language Environment) generation.

Externalizing the Function Interface

When the interface to a subroutine is inside the subroutine, each time it is called a new version of the subroutine is required. To externalize the function interface, a unique internal work field is assigned for each parameter field and the interface is generated as Move statements before and after the subroutine call. The names of the internal work context fields are generated in a similar fashion to that of Neither parameters. This provides up to 10,000 unique field names in a given program.

When an internal function is called:

1. The function's Input and Both parameters are moved to the new internal work context prior to the call.
2. Within the called function, this internal work context is used instead of the original PAR context.
3. Output and Both parameters are moved back to the return context after the subroutine call.

Note: The Moves required before and after calling a subroutine increase overhead somewhat. If a function has more parameters than executable statements, then reusing the subroutine increases the number of source lines generated.

You control the sharing of subroutines using the Share Subroutine (YSHRSBR) model value and its associated function option. The table shows the valid values; function option values are shown in parentheses. The default is *NO.

Value	Description
*YES (Y)	Share generated source for subroutines. Generate source code the first time an internal function is called and reuse the source for all subsequent calls to the function. The interface for the subroutine is externalized.
*NO (N)	Generate source code each time an internal function is called. The interface for the subroutine is internal.

The YSHRSBR model value and function option are available on the CHGOBJ, CRTOBJ, DLTOBJ, RTVOBJ, and EXCINTFUN function types.

In addition, the Generate as subroutine? function option is provided for the EXCINTFUN function type to indicate whether to implement the function in-line or as a subroutine. The default is not to generate as a subroutine.

Using Shared Subroutines with EDTFIL, EDTTRN, EDTRCD

By default, the EDTFIL, EDTTRN, and EDTRCD(n) functions contain a call to the CHGOBJ and DLTOBJ functions of the owning file. When they are generated as subroutines, they include a section of code that checks whether the record about to be deleted or changed has been changed by another user since being displayed to the screen. However, if the same CHGOBJ or DLTOBJ functions are inserted elsewhere in the action diagram of an EDTxxx function, this code is not generated. Consequently, if the CHGOBJ or DLTOBJ functions are marked for sharing, any EDTxxx function that contains both the default instance and more than one further instance of that internal function generates two separate subroutines—one used only at the default user point containing the previous code and one for use at all other points that does not contain this code.

Understanding the Action Diagram Editor

The action diagram editor lets you modify the default processing logic that is automatically supplied for a function. It also provides the ability to add, change, or delete actions at appropriate points in the structure of a function. These points are called user points.

Selecting Context

To select a context use the steps in the previous topic, Defining Compound Conditions, and the following steps.

1. From the Edit Action - Condition panel enter the context you want to select or a question mark ? in the Context field and press Enter.

   The Display Field Contexts panel appears.

2. Depending on the type of function you are editing and the point in the action diagram, select the context.

Entering and Editing Field Conditions

With the action diagram editor you can add, modify, and delete conditions.

Adding Conditions

To add conditions use the following steps:

1. From the Edit Action - Condition panel type **?** in the Condition field and press Enter.

   Or do the following:

   From the Edit Database Relations panel type **Z2** next to the file to field relation for which you want to define conditions.

   The Edit Field Details panel appears.

2. From this panel, change any of the field attributes and add any narrative text.
3. Press F9 to view the field conditions.

   The Edit Field Conditions panel appears.

4. Enter a new condition. Type the name of the condition in the Enter Condition field and the condition type, VAL or LST for status fields or CMP and RNG for non-status fields and type **Enter**.

   The Edit Field Condition Details appears.

5. If your field is of type status, type the internal file value associated with the condition name. For instance, P for the field condition Paid.
6. If your field is of any type other than status and the condition type is range (RNG), type the From and To range.
7. If your condition type is Compare (CMP), type the comparison operator (EQ, GT) and the comparison value.

Deleting Conditions

To delete conditions, use the following steps:

1. From the Edit Field Conditions panel, type **D** next to the condition that you want to delete and press Enter.

   Note: If the field condition is used in the function logic processing of any function, you are not able to delete it until you resolve the function references.

2. Press U to view a list of references for your field condition.

Line Commands

The line commands available for use with the action diagram editor appear above the panel subfile and are listed and described next and on the following pages. You can prompt for the complete list of line commands by typing ? in the line command positioner field. The F as a suffix on the command prefills any fields with question marks for prompting.

I (Insert)

There are several insert line (I) commands that you can use in the action diagram editor to insert constructs. You can use the insert line command to insert constructs in action diagram shells or even within other constructs.

- **IA (IAF)**—Inserts an action within a construct or in an action diagram shell.
- **IC (ICF)**—Inserts a condition within a CASE construct.
- **IE(IEF)**—Inserts a *EXIT PROGRAM built-in function.
- **II (IIF)**—Inserts an iterative construct within a REPEAT WHILE construct.
- **IS (ISF)**—Inserts a blank sequential construct.
- **I* (I*F)**—Inserts a comment at any point within the user point.
- **IM (IMF)**—Calls a message function at a particular point within the action diagram. When you enter **IM** in the action diagram user point, the Edit Message Functions panel appears.
- **IO**—Inserts an otherwise clause.
- **IQ(IQF)**—Inserts a *QUIT built-in function.
- **IX (IXF)**—Inserts a new condition within the CASE construct.
- **I+ (I+F)**—Calls the *ADD (add) built-in function. This takes you to the Edit Action - Function Details panel.
- **(I-F)**—Calls the *SUB (subtract) built-in function. This takes you to the Edit Action - Function Details panel.
- **I= (I=F)**—Calls the *MOVE (move) built-in function. This takes you to the Edit Action - Function Details panel.
- **I=A**—Calls the *MOVE ALL (move all) built-in function. This takes you to the Edit Action - Function Details panel. You can also use I = = .
- **I=M**—Calls the *MOVE ARRAY (move array subfield) built-in function. This takes you to the Edit Action - Function Details panel.

M or MM (Move) (A or B)

The move (M) line command allows you to move a construct to a point that you designate within the action diagram shell, either A (after) or B (before).

The move block (MM) line command allows you to move a block of constructs to a point that you designate within the action diagram shell, either A (after) or B (before). The MM line command must be paired with another MM line command at the same construct level.

This command does not edit the field context for the new user point.

C or CC (Copy) (A or B)

The copy (C) line command allows you to copy a construct to a point that you designate within the action diagram shell, either A (after) or B (before).

The copy block (CC) line command allows you to copy a block of constructs to a point that you designate within the action diagram shell, either A (after) or B (before). The CC line command must be paired with another CC line command at the same construct level.

This command does not edit the field context for the new user point.

D or DD (Delete)

The delete (D) line command allows you to delete a construct.

The delete block (DD) line command allows you to delete a block of constructs. The DD line command must be paired with another DD line command at the same construct level.

N (Narrative)

The narrative (N) line command lets you edit the object narrative for the selected function or message.

PR (Protect)

The protect action diagram block (PR) line command lets you protect a selected action diagram block. Requires *DSNR with locks authority.

For more information on protecting action diagram blocks, see this chapter, Protecting Action Diagram Blocks.

R (References)

The references (R) line command displays references for the function or message referenced by the selected action diagram entry. For functions, references are expanded to the first external function; for messages, references are expanded to the next level. Note that changes to the action diagram are not reflected in the references until the function is updated.

U (Usages)

The usages (U) line command displays usages for the function or message referenced by the selected action diagram entry. Note that changes to the action diagram are not reflected in the usages until the function is updated.

V (View Summary)

The view summary (V) line command displays a summary of selected block.

S (Show)

The show (S) line command allows you to reverse the effect of hiding a construct.

H (Hide)

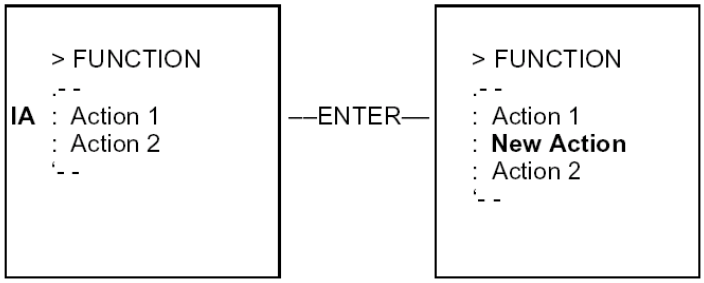
The hide (H) line command allows you to hide a construct. The construct executes in the normal fashion. However, instead of displaying all lines in the construct, only one line displays indicating that the construct has been hidden.

Z (Zoom)

The zoom (Z) line command allows you to focus in on a particular construct and display all ancillary parts of the construct. This command also allows you to navigate your way through embedded constructs or into an action diagram of an embedded function.

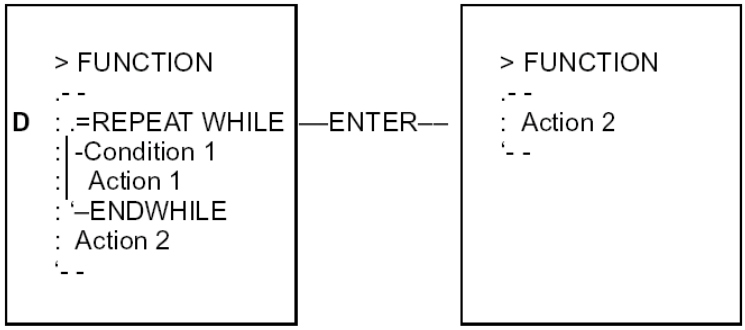
Adding an Action —IA Command

The following example shows the effect of adding a new action (IA):



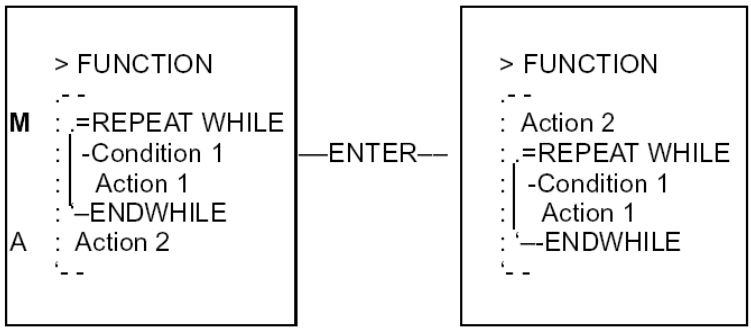
Deleting Constructs—D Line Commands

Non-protected constructs can be deleted by placing a D against the line.



Moving a Construct—M and A Line Commands

Non-protected constructs can be moved from one position in the action diagram to another. To move a construct, place an M against the construct that is to be moved, and an A against the line of the action diagram after which the construct is to be moved.



Function Keys

The following is a list of function keys that are used within the Action Diagram Editor.

Function	Description
F3	Returns the cursor to the position of the previous zoom. If no previous zoom, exits the action diagram.
F5	Display user points
F6	Cancel pending move operations

F7	Scan backward
F8	Creates a bookmark for the current cursor location in the action diagram and adds it to the list of bookmarks. See F20.
F9	Edit Function Parameters
F12	Enzyme, one block at a time.
F13	Exit the action diagram
F14	Display CA 2E Map
F15	Open Function Panel
F16	Toggle change date
F17	Display Action Diagram Services panel to search for function, field, change date or any syntax error found in the action diagram.
F18	Access or leave Notepad
F19	Edit device design
F20	Display bookmarks. Select a bookmark to quickly position to that point in the action diagram.
F21	Toggle implementation names and function types.
F23	View more line commands
F24	View more command keys.
ENTER	Execute line commands
HELP	Display help text
ROLLUP	Show next page of work area
ROLLDOWN	Show previous page of work area

Using NOTEPAD

The notepad utility allows you to copy constructs from one action diagram to another. When using Open Functions, this utility lets you save the contents of a diagram to a work area and copy those contents elsewhere.

NOTEPAD Line Commands

The Notepad line commands are described next.

NI (NOTEPAD Insert)

The notepad insert (NI) allows you to insert the contents of a notepad at a point after the line on which the cursor is positioned.

NA or NAA (NOTEPAD Append)

The notepad append (NA) line command allows you to copy the contents of a construct to a notepad and to append them to the existing contents of the notepad.

The notepad block append (NAA) line command allows you to copy the contents of a block of constructs to a notepad and to append them to the existing contents of the notepad. The NAA line command must be paired with another NAA line command at the same construct level.

NR or NRR (NOTEPAD Replace)

The notepad replace (NR) line command allows you to first clear the contents of the notepad, and then to replace the existing contents of the notepad with a new construct.

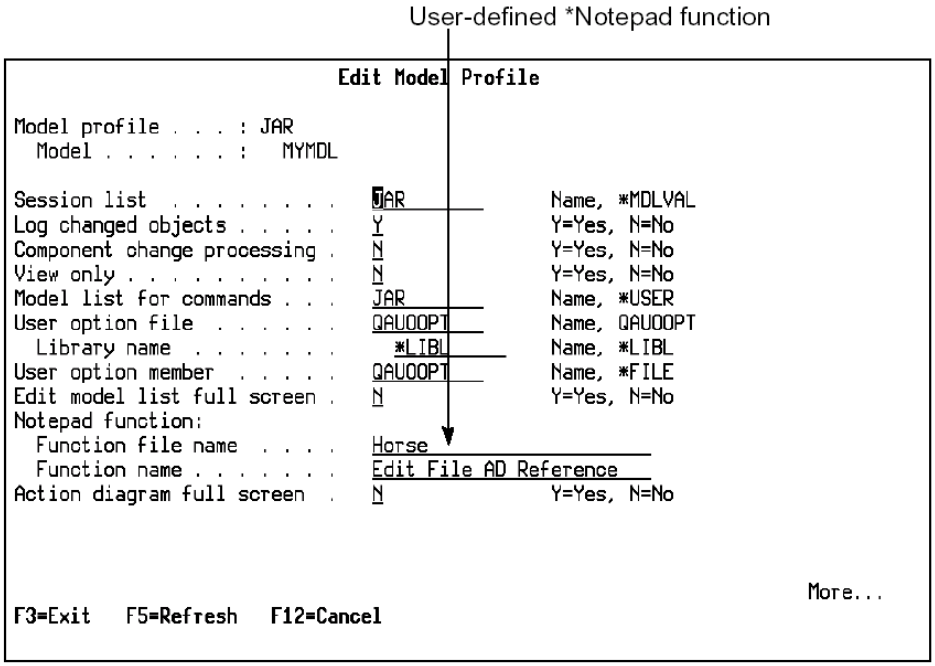
The notepad block replace (NRR) line command allows you to first clear the contents of the notepad, and then to replace the existing contents of the notepad with a new block of constructs. The NRR line command must be paired with another NRR line command at the same construct level.

You can toggle between the notepad action diagram from the action diagram you are editing by pressing F18**.**

User-Defined *Notepad Function

You can specify the file and name of a user-defined *Notepad function in the model profile. This function can be either an EXCINTFUN or EXCEXTFUN function type. A user-defined *Notepad function can serve as a repository of standardized action diagram constructs that you can easily copy into the action diagrams of other functions.

Use the Edit Model Profile (YEDTMDLPRF) command or the Edit Model Profile option on the Display Services Menu to specify a user-defined *Notepad function.



If you do not specify a user-defined *Notepad function, the shipped non-permanent *Notepad function is used by default. The shipped *Notepad function is also used if the function specified in the model profile does not exist or if you select it as the primary function to be edited.

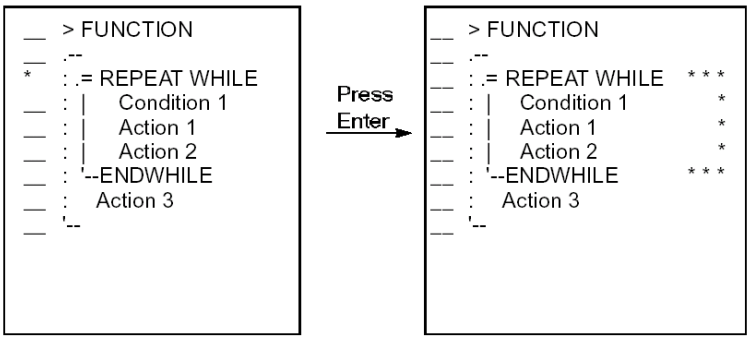
Multiple developers may use the same user-defined *Notepad function. The first developer to access the function has an update lock on it and can update it on exit.

*, ** (Activate/Deactivate)

The activate/deactivate line command (*,**) allows you to deactivate or activate a construct or block of constructs. The ***** line command is used to toggle the active/deactive flag for a construct. Deactivated constructs do not generate any associated code, nor does any otherwise active construct that is nested. Deactivating a construct is similar to wrapping an always false CASE structure around the construct. The action of the * line command depends on the status of the construct.

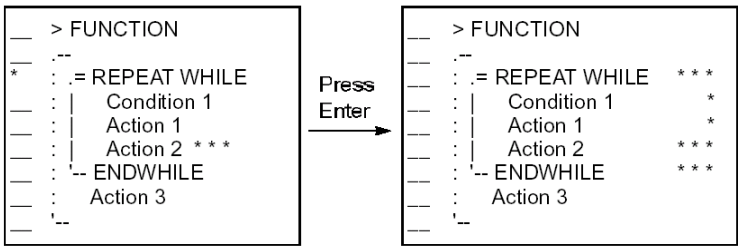
If the construct is currently active, * deactivates that construct. This then displays using the attributes of COLOR(WHT), DSPATR(HI). In addition, deactivated constructs have a *** symbol displayed to the right of the action diagram line.

If the construct is currently deactivated, * reactivates that construct. The method of display is now dependent on the activation status of its parent constructs. If any of its parent constructs (within which it is nested) are currently deactivated, the construct still displays as if it were deactivated but the symbol on the right will be a *. This indicates that the construct has inherited the deactivation status of its parent construct.

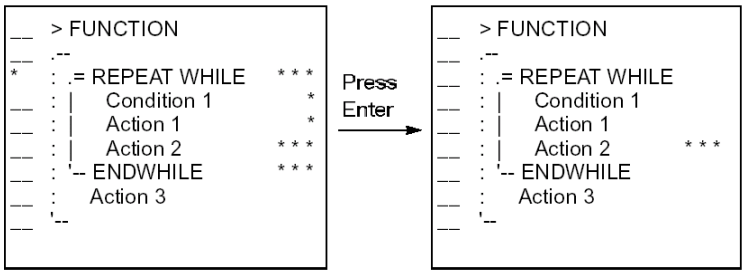


Note: The display of the * symbol for inherited deactivation is preserved for zooms into hidden structures within the same action diagram. However, it is not preserved for zooms into other action diagrams. These are deactivated only for use within the parent function. They are not generated within that function, but because they are not inherently deactivated, the deactivation is not indicated while editing them.

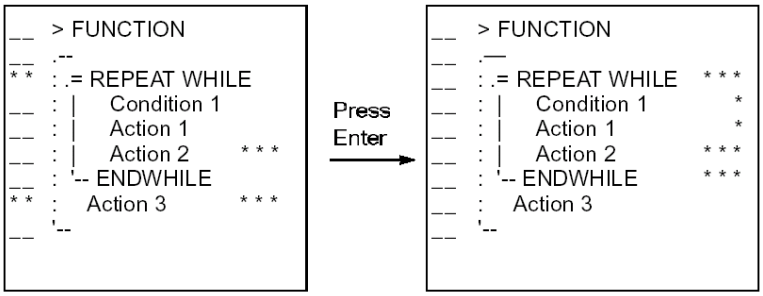
If a nested construct is currently deactivated when its parent construct becomes deactivated, its own deactivation status is not changed and it remains deactivated. If it is currently active, it inherits the deactivation status of its parent.



If a nested construct is currently deactivated when its parent construct becomes reactivated, its own deactivation status is not changed and it remains deactivated. If it is currently active, it no longer inherits the deactivation status of its parent and it is reactivated.



If a pair of **s is used, these must be defined at the same construct level. Each construct at the same level as the ** has its associated active/deactive flag toggled. This can lead to some constructs being deactivated and some being reactivated. Constructs nested within these constructs are not updated but still inherit the deactivation of their parent constructs.



Protecting Action Diagram Blocks

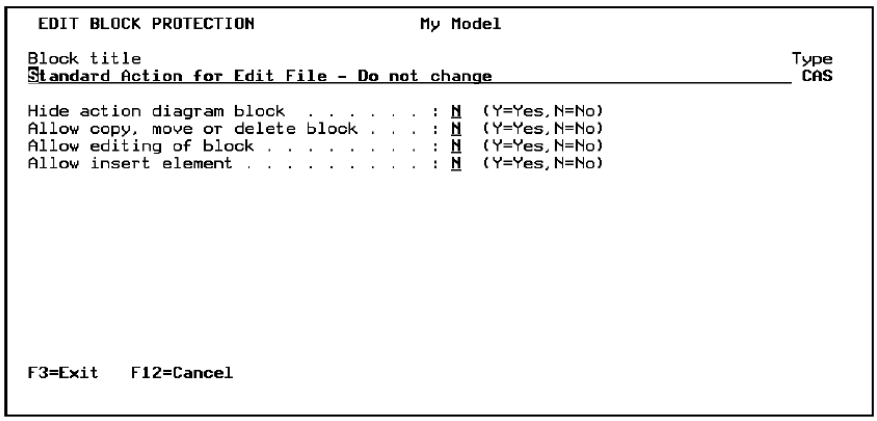
This feature lets a *DSNR with locks capability (*ALL authority to YMDLLIBRFA) prevent all developers from editing, copying, moving, deleting, or inserting statements within the protected action diagram block. You can protect a single action, a case block, an iteration block, a sequence block, or a comment.

An important use for this capability is to protect standardized areas in the action diagrams of your function templates.

For more information on function templates, see this module, Chapter 8, "Copying Functions," Template Functions.

Protecting a Block

In the action diagram, type **PR** against the block you want protected. The Edit Block Protection panel appears for the selected construct or block.



Note that the type of the block or construct appears; in this example, it is CAS for case block. The type can be ACT, CAS, ITR, SEQ, or TXT.

1. If you specify a Block title, it displays in the action diagram for case, iteration, and sequence blocks. You can specify up to 74 characters. The Block title does not display for actions and comments; however, you can search for Block title text for all protected blocks using the Action Diagram Services panel.
2. Specify the type of protection you want the block to have:

Protection Type	Description	Valid Blocks
Hide	If Y, the protected block is hidden.	Case Iteration Sequence
Allow Copy, Move or Delete	If N, developers are prevented from copying, moving, or deleting the block.	All
Allow Edit	If N, developers are prevented from editing or inserting a block.	All
Allow Insert	If N, developers are prevented from inserting blocks within the protected block. **Note:** To insert a block, both Allow Insert and Allow Edit must be set to Y.	Case Iteration Sequence

3. Press Enter and then F3 to exit.

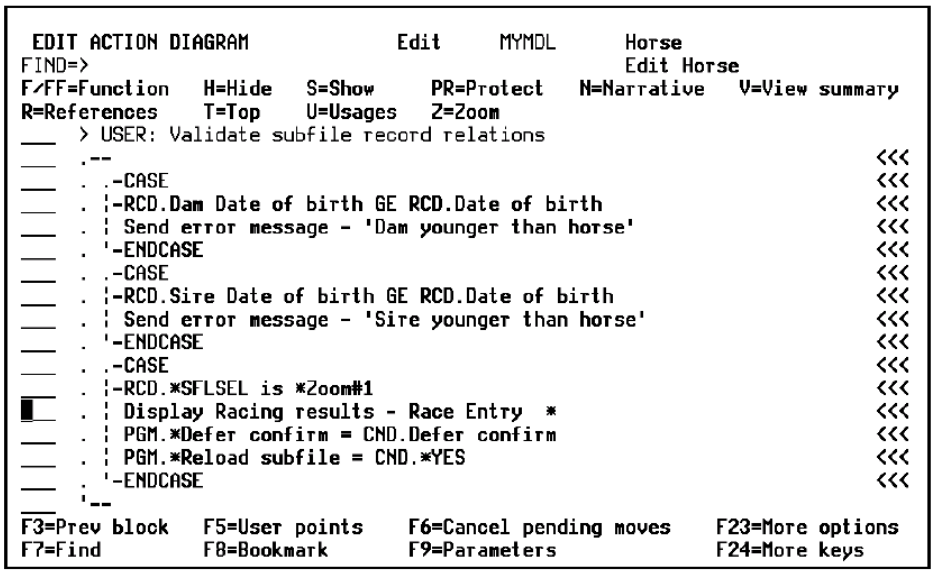
Note: Only the specified block is protected, not blocks embedded within.

Protected blocks appear as if they were part of the action diagram prototype.

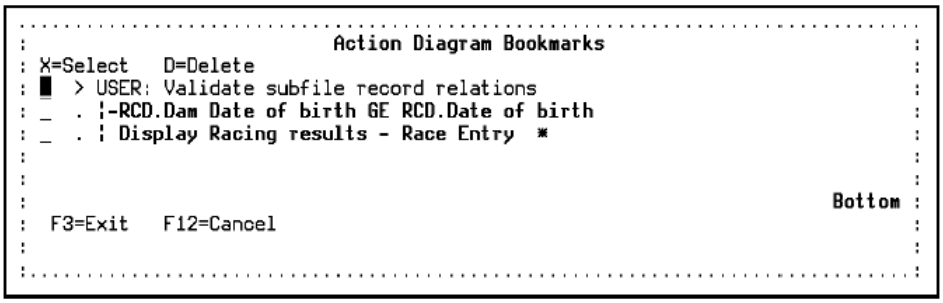
Using Bookmarks

A bookmark is a record of a location in the action diagram that you can use later to quickly return to the marked location. You can create any number of bookmarks in an action diagram.

To create a bookmark, place the cursor on the line in the action diagram where you want the bookmark and press F8.

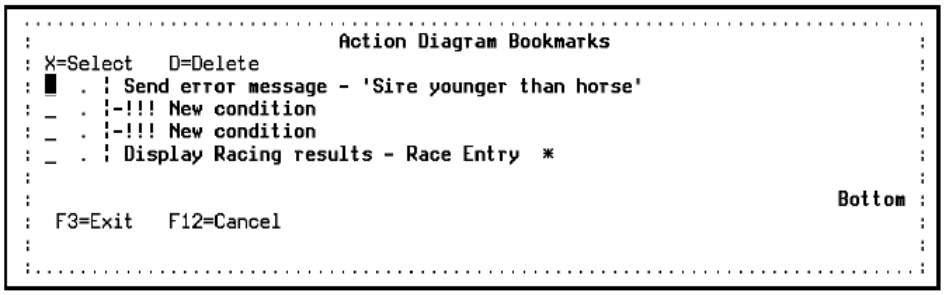


Each bookmark you create is added to a list of bookmarks. Press F20 to display a list of the existing bookmarks for the action diagram.

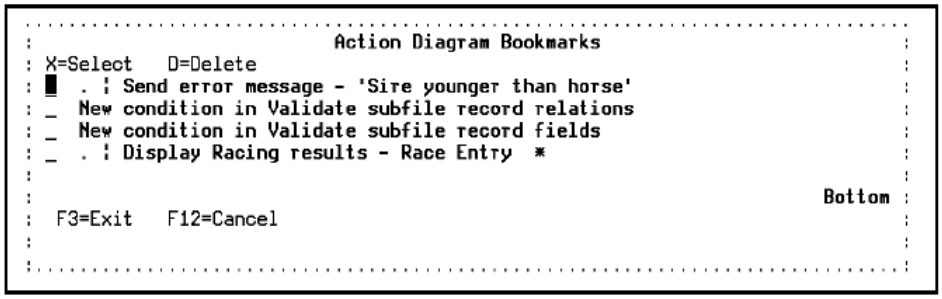


Type **X** against a bookmark to select it and return to the marked location in the action diagram; type **D** against a bookmark to delete it. Press Enter.

By default, a bookmark is identified on the bookmark list by its text from the action diagram. This can result in similar or identical entries in the bookmark list. For example,



To distinguish such entries, edit the text by typing over the existing bookmark text. For example,



If you delete the action diagram entry associated with a bookmark, the bookmark is deleted. Otherwise, changes to the action diagram are not reflected in the bookmark list.

You can maintain a separate list of bookmarks for each open function.

You can choose to save bookmarks when you exit a function. To do this, ensure that the option on the EXIT FUNCTION DEFINITION panel called Save bookmarks is set to 'Y AND the Change/Create Function flag is set to Y. If you don't set these flags to Y on exit, then the Bookmark list disappears when you exit the function

Submitting Jobs Within an Action Diagram

This feature lets you specify within the action diagram that a function is to be submitted for execution in batch using the Submit Job (SBMJOB) command. You can override the SBMJOB command parameter defaults at the model, function, or action diagram level or dynamically at run time. In addition, references to the submitted functions are visible using CA 2E's impact analysis facilities.

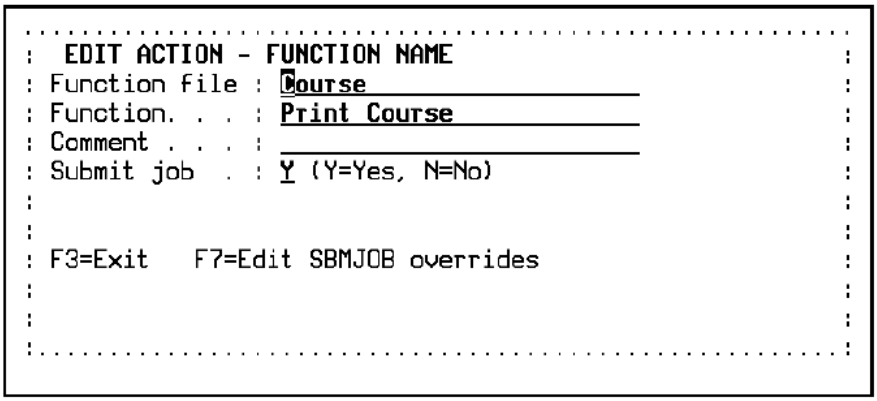
Notes:

- Only EXCEXTFUN, EXCUSRPGM, and PRTFIL functions can be submitted for batch execution using this method.
- This feature does not support function calls that contain multiple-instance array parameters.

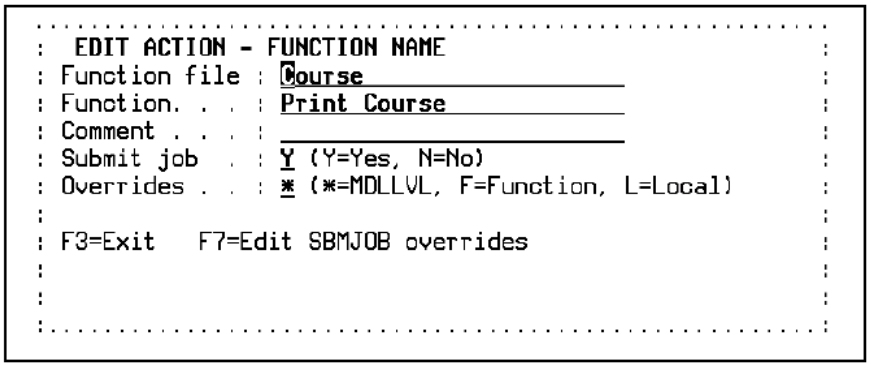
Inserting a SBMJOB in an Action Diagram

1. Go to the location in the action diagram of the function where the SBMJOB is to be inserted and type **IAF**. Press Enter to display the Edit Action - Function Name window.

   Type the file and name of the function to be submitted for batch execution and press Enter. If the function you specified is an EXCEXTFUN, EXCUSRPGM, or PRTFIL function, the Submit job option appears. Type **Y** to indicate that the function is to be submitted for batch execution.



   Press Enter to display the Overrides option.



   The Overrides option specifies the source of the SBMJOB parameter overrides to use for this call.

Value	Description
*	Model level
F	Function level
L	Action diagram (local) level

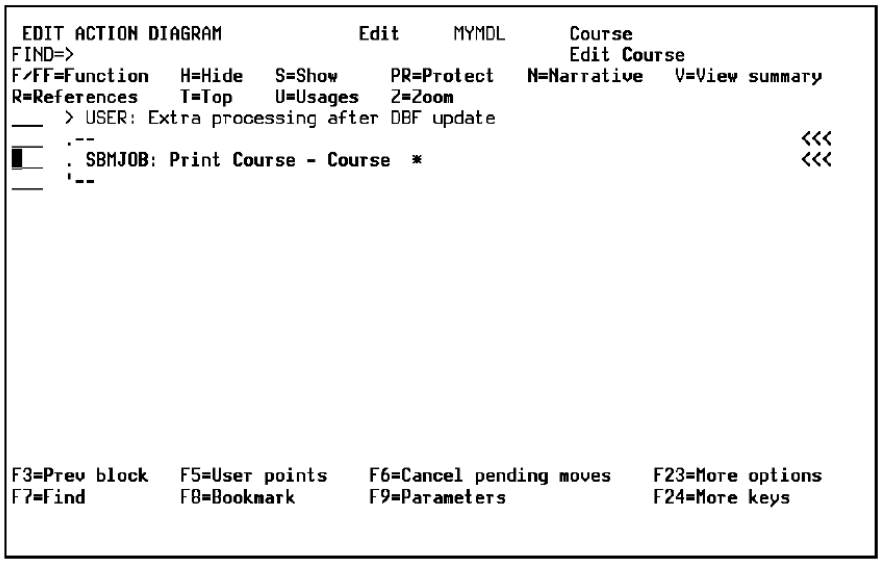
Type the appropriate value.

2. If you specified L for Overrides, press F7 to specify local SBMJOB command parameter overrides for this function. When finished, press Enter.
3. Specify parameters for the referenced function. Each parameter is formatted and added to a command string that is ultimately invoked as the CMD parameter of the Submit Job (SBMJOB) command.

   Note: Performance degrades if you specify many individual parameters on the call. As a result, it is recommended that you use RCD or KEY parameters only, rather than FLD specifications.

   Press Enter.

   The action diagram displays with the completed action.



   Note: The SBMJOB: prefix indicates that the referenced function will be submitted for batch execution.

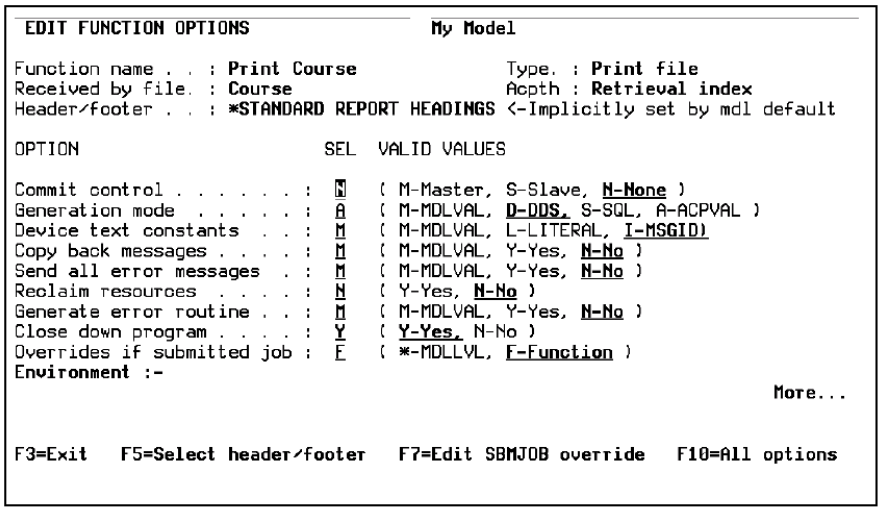
4. Complete your work on the action diagram and save the function.

Defining SBMJOB Parameter Overrides

You can override the parameters of the Submit Job (SBMJOB) command at three levels at source generation time and dynamically at run time. You can override any parameter except CMD.

Source Generation Overrides

- **Model level**—The model level override is stored in a system supplied message called *Sbmjob default override attached to the *Messages file. Edit this message to define model level overrides. Function and local overrides default to the model level.
- **Function Level**—The F7 function key on the Edit Function Options panel lets you edit parameters for the SBMJOB command at either the model or the function level.



The Overrides if submitted job function option specifies the source of the SBMJOB parameter overrides and which level of overrides you are editing when you press F7. The values are:

Value	Description
*	Model level. Use the default overrides defined by the *Sbmjob default override message attached to the *Messages file in Y2USRMSG. Press F7 to edit the model level overrides.
F	Function level. Use the override defined for this function. You define or edit the function level default using F7 on the Edit Function Options panel. If not specified, it defaults to the model level override

To specify a function level override, set the function option to F and press F7 to edit the SBMJOB command parameters.

Note: You also need to specify F for the Overrides option on the Edit Action - Function name window in the action diagram for the function level overrides to take affect. You can also edit the function level overrides at that point.

- **Action Diagram (Local) Level**—The Edit Action - Function name window provides two additional options for EXCEXTFUN, EXCUSRPGM, and PRTFIL functions:
  - Submit job indicates whether the referenced function is to be submitted for batch execution:

Value	Description
Y	Submit the function to batch
N	Process the function interactively

  - Overrides specifies both the source of SBMJOB command parameter overrides and which override level you are editing when you press F7. Note that this parameter displays only if Submit job is set to Y.

Value	Description
*	Model level override. Use the model level override string stored in the *Sbmjob default override message. Press F7 to edit the model level overrides.
F	Function level override. Use the override specified by the Overrides if submitted job function option for the referenced function. You can edit the function option overrides by pressing F7.
L	Action diagram level override. Use the override that has been defined for this call. Press F7 to enter or edit local overrides.

To specify an action diagram level override set the Overrides option to L and press F7 to edit the SBMJOB command parameters.

Note: Local overrides do not update the action diagram until you save the function on exit. Function and model level overrides are updated immediately. If later you change the function that is to be submitted to batch, any local SBMJOB parameter overrides you previously specified are retained.

Dynamic Overrides

To provide a dynamic override to SBMJOB command parameters at run time, move the keywords and parameter values into the new PGM context field *Sbmjob override string. If this field is not blank at run time its contents are merged into the *Sbmjob default override message, overriding any existing values.

The *Sbmjob job name, *Sbmjob job user, and *Sbmjob job number PGM context fields facilitate additional processing for the submitted job; for example, handling spool files, follow-on updates, lock manipulation, and any other processing that requires submitted job information.

Special SBMJOB Considerations

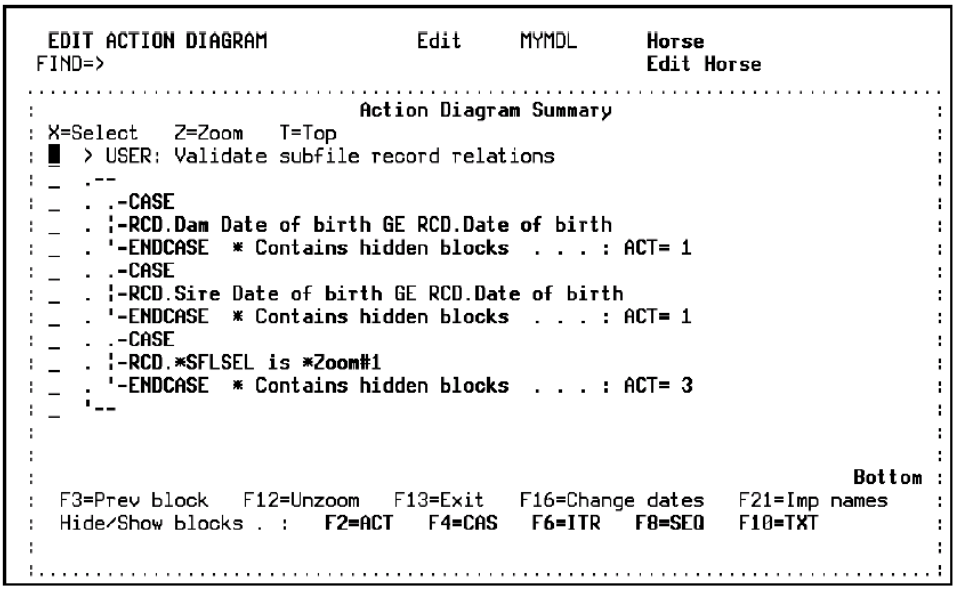
Advantage of SBMJOB Over Execute Message

Some advantages of using the SBMJOB feature over Execute Message to submit commands or programs for batch execution are:

- Numeric parameters can be passed.
- The complexities of constructing the submit job command string are hidden.
- References to submitted functions are visible by CA 2E impact analysis facilities.

Viewing a Summary of a Selected Block

Type **V** against an action diagram block to view a summary of its contents. The Action Diagram Summary window appears:



Note: You can also access the Action Diagram Summary window using the V option on the User Exit Points window.

The summary lets you quickly select a portion of a large or complicated action diagram for editing. Within the summary, you can easily zoom to a lower level (Z) and select a block for editing (X).

The action diagram is summarized by hiding combinations of blocks. The last line of the summary construct shows the type and number of blocks contained within that portion of the action diagram; for example, the following indicates that the Case block contains three (ACT=3) actions:

```
_  . .-CASE
_  . |-RCD.*SFLSEL is *Zoom#1
_  . '-ENDCASE  * Contains hidden blocks  . . . : ACT= 3
_  '--
```

Use the following function keys to hide or show selected types of blocks

Function Key	Description
F2	Hide and show actions
F4	Hide and show Case blocks
F6	Hide and show Iteration blocks
F8	Hide and show Sequence blocks
F10	Hide and show Comments

Using Action Diagram Services

Action Diagram Services lets you perform searches on functions, files, or fields in the Action Diagram Editor. This utility also allows you to search according to date and for any constructs that contain errors. You can specify the direction of the scan, the location where the scan is to start, and the amount of detail to display for a successful scan.

In addition, you can use this panel to toggle full screen mode in the action diagram and call functions from within an action diagram.

Scanning for Specified Criteria or Errors

Access Action Diagram Services by pressing F17 from the Edit Action Diagram panel. The Action Diagram Services panel appears.

```
 Action Diagram Services                              SYNMDL


Type choices, press ENTER.

  Find option  . . . . . .  :   1 1=Criteria, 2=Error

  Occurrences to process . .  :   1 1=Next, 2=All, 3=Previous

  Display find in context  .  :   2 1=Exact, 2=Block, 3=User point

  Find function file name  .  :               Name, *ALL

  Find function name . . . . .:               Name, *ALL

  Find field name  . . . . .. :               Name, *ALL

                 Contex. . . :       Usage . .:   (I/O/B/U)

  Search for date  . . . . :  :  5/06/13  CYYMMDD

  Compare  . . . . . .  :     1=Less than, 2=Less than or equal to,

                              3=Equal to,  4=Greater than or equal to,

                              5=Greater than

  Scan in titles and comments .:                            Ignore case . : Y

  Scan for implementation name :

  Reset change dates . . . .   :     (Y/N)          Share find criteria . . : N

  Full screen mode  . . . . .  :  N  (Y/N)          Auto-scan functions . . : N

  Call function  . . . . . . :


F3=Exit   F7=Find   F9=Command line   F11=Conditions   F12=Cancel   F16=Y2CALL
```

1. To perform a search for specified find criteria, type option **1** in the Find Option field.

   To specify the criteria of the search:

   a. In the Find Function File Name field, type the name of the file, *ALL if you want to search all files, or ? if you are uncertain of the file name.

b. Type the name of the field in the Find Field Name field. To perform a search on the field and/or context or usage, enter the field name and the context type. You can enter the field name without entering the context; however, you cannot enter the context or usage without specifying a field name.

c. Enter the date in YYMMDD format in the Search for Date field.

d. For a specific date comparison, select one of the options for the following criteria:

- Less than (for a date prior to the date).
- Less than or equal to (for a date prior or equal to the search date)
- Equal to (for a date equal to the search date)
- Greater than or equal to (for a date after or equal to the search date)
- Greater than (for a date after the search date)

a. To scan only in block titles and comments, type the text to scan for in the Scan in titles and comments option. The wildcard character ? indicates a generic scan; a leading ? is not valid.

b. To search functions and messages for a specified source member name or message identifier respectively, type the implementation name for the Scan for implementation name option. To make the resulting action diagram display clearer, press the F21 key when the search is successful.

1. To perform a search for syntax errors in the action diagram, from the Action Diagram Services panel, type **2** for the Find option.
2. To search for matches to the specified criteria or errors in the action diagram, press F7 to scan forward. To scan backwards, type **3** for the Occurrences to process option.
3. Share find criteria is a Y or N option specifying whether the find criteria entered on this screen are shared between open functions. If set to N, each open function uses its own find criteria, which are initialized when the function is first opened and are retained while the function remains open.

   If set to Y, all open functions share a single set of find criteria. Changes made to the find criteria (using this screen) when accessed from one open function are retained and used for all other open functions.

   N Open functions each use their own set of find criteria.

   Y Open functions share a single set of find criteria.

   Note: If this screen is accessed from a function and this value is set to Y, and another function is subsequently opened, you can use F7 (Scan) immediately in that function without the need to re-access this screen.

4. Auto-scan functions: This flag is only valid if Share find criteria is set to Y. This flag determines whether the specified scan should be performed on a function as soon as the function is edited. This flag is automatically set to Y if the F7 key is pressed to exit this program and this program was called from the OPEN FUNCTIONS panel. The autoscan functionality applies to any open functions, including the function being edited when this program was called (if any) as well as to any functions that are subsequently zoomed into.

   N The specified scan will not be performed automatically on return from this program—the user must press F7 when editing the action diagram of the function to start the scan.

   Y The specified scan is automatically performed on return from this program.

Note: Error checking is also available outside Action Diagram Services using the Check Function Action Diagram (YCHKFUNACT) command or option 38 on the YEDTMDLLST panel. If any errors are found, the action diagram is loaded and positioned to the first error.

Calling Functions Within an Action Diagram

The Call function option lets you call an external function from within an action diagram.

Calling an External Function

1. Type the name of the function to be invoked on the first line of the Call function option. This defaults to the function you are editing.
2. Type the name of the file the function is attached to on the second line. Press Enter.
3. Press F16 to invoke the Call a Program (Y2CALL) command. Adjust any parameters and press Enter to invoke the call.

Calling an Internal Function

Since an internal function cannot be called directly, you need to select an external function that calls it.

Type **?** for the Call function option and press Enter. The Select External Function panel displays showing usages for the function you are editing. External functions are highlighted.

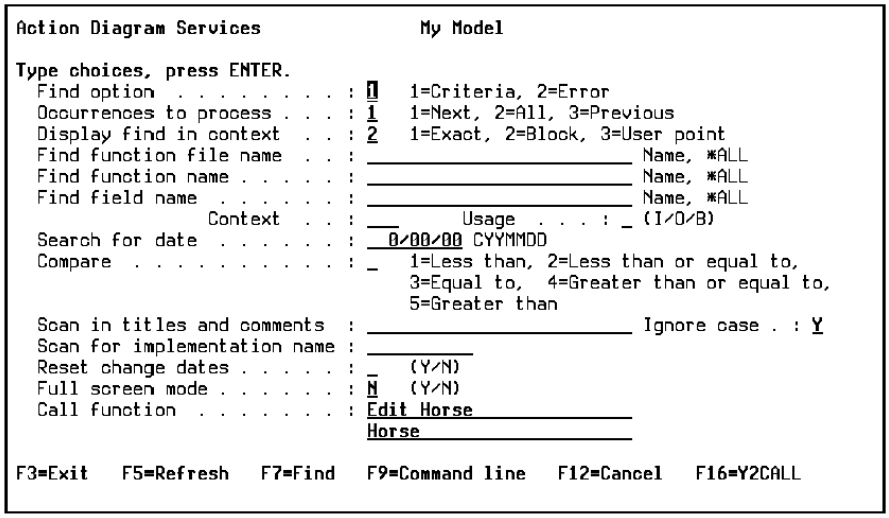
```
 Action Diagram Services                                SYNMDL


Type choices, press ENTER.

  Find option  . . . . . :  1  1=Criteria, 2=Error

  Occurrences to process . . . . :  1   1=Next, 2=All, 3=Previous

  Display find in context  . . . :  2   1=Exact, 2=Block, 3=User point

  Find function file name  . . . :               Name, *ALL

  Find function name . . . . . :               Name, *ALL

  Find field name  . . . . . :               Name, *ALL

                Context  . . . :      Usage  . . . :   (I/O/B/U)

  Search for date  . . . . . :  5/06/13  CYYMMDD

  Compare  . . . . . :     1=Less than, 2=Less than or equal to,

                                    3=Equal to,  4=Greater than or equal to,

                                    5=Greater than

  Scan in titles and comments .  :                Ignore case . : Y

  Scan for implementation name   :

  Reset change dates . . . . . :    (Y/N)        Share find criteria . . : N

  Full screen mode  . . . . . :  N  (Y/N)        Auto-scan functions . . : N

  Call function  . . . . . :  Edit Horse

                                  Horse


F3=Exit   F7=Find   F9=Command line   F11=Conditions   F12=Cancel   F16=Y2CALL
```

1. If the display of usages is extensive, use the function keys to position the display to the appropriate function.

   For more information on use of the function keys for this panel, see the online help.

2. Use the selection options to select a function from one of the three displayed columns, which are numbered from left to right. Press Enter.

   The Action Diagram Services panel redisplays with the selected function's name and file displayed for the Call function option.



3. Press F16 to prompt the Call a Program (Y2CALL) command. Adjust any parameters and press Enter to invoke the call.

Additional Action Diagram Editor Facilities

CA 2E makes the following additional facilities available to you while you are in the action diagram.

Editing the Parameters

To modify your current action diagram parameters, press F9 from the Edit Action Diagram panel. CA 2E displays the Edit Function Parameters panel. To return to the action diagram press F3.

For more information on modifying parameters, see Chapter 5, "Modifying Function Parameters."

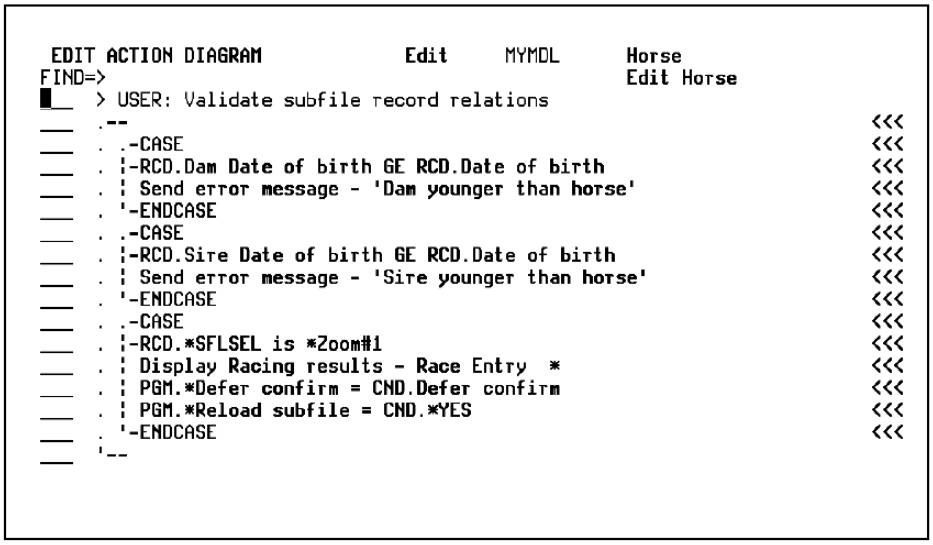
Toggling to Device Designs

To toggle to the device design associated with the current action diagram, press F19. To return to the action diagram, press F3 and then F5.

Full Screen Mode

In full screen mode, no subfile option or function keys display on the Edit Action Diagram panel and the subfile page is expanded to fill the space. The Action diagram full screen option in the model profile sets the default mode. You can override this value for any function using the Full screen mode option on the Action Diagram Services panel. Use this option to return to normal mode.

Following is an example of the action diagram in full screen mode:



Toggling Display for Functions and Messages

The F21 function key lets you toggle the information displayed for functions and messages. For functions, the implementation name and function type display; for messages, the message id and message type display.

For example, the following is the default display for a function.

```
.-CASE                                                              <<<
|-RCD.*SFLSEL is *Zoom#1                                            <<<
| Display Racing results - Race Entry  *     MYALDFR                <<<
| PGM.*Defer confirm = CND.Defer confirm                            <<<
| PGM.*Reload subfile = CND.*YES                                    <<<
'-ENDCASE                                                           <<<
```

Press F21 a second time to display the function type:

```
  Action Diagram Services                                    SYNMDL

 Type choices, press ENTER.
   Find option  . . . . . . . . . . .: 1  1=Criteria, 2=Error
   Occurrences to process . . . . . .: 1  1=Next, 2=All, 3=Previous
   Display find in context  . . . . . : 2   1=Exact, 2=Block, 3=User point
   Find function file name  . . . . . :              Name, *ALL
   Find function name . . . . . . . . :              Name, *ALL
   Find field name  . . . . . . . . . :              Name, *ALL
                  Context  . . . . . :      Usage  . . . :   (I/O/B/U)
   Search for date  . . . . . . . . . .: 5/06/13  CYYMMDD
   Compare  . . . . . . . . . . . . . :    1=Less than, 2=Less than or equal to,
                                           3=Equal to,4=Greater than or equal to,
                                           5=Greater than
   Scan in titles and comments  :                            Ignore case . : Y
   Scan for implementation name :
   Reset change dates . . . . . :      (Y/N)         Share find criteria . . : N
   Full screen mode  . . . . . .:  N   (Y/N)         Auto-scan functions . . : N
   Call function  . . . . . . . :

  F3=Exit    F7=Find    F9=Command line    F11=Conditions    F12=Cancel    F16=Y2CALL
 Editing and Maintaining Functions Simultaneously
```

You have the ability to open, edit, and maintain several functions simultaneously.

You have the ability to open, edit, and maintain several functions simultaneously.

Starting Edits for Multiple Functions

To open multiple functions for a file simultaneously, use either of the following panels.

- **Edit Functions panel**—Enter **O** for each of the functions you want loaded.

  Note: If you enter O for a function on the Edit Functions panel, any subsequent F or S subfile select option you enter is interpreted as O.

- **Edit Model Object Lists panel**—Enter **30** for each of the functions you want loaded.

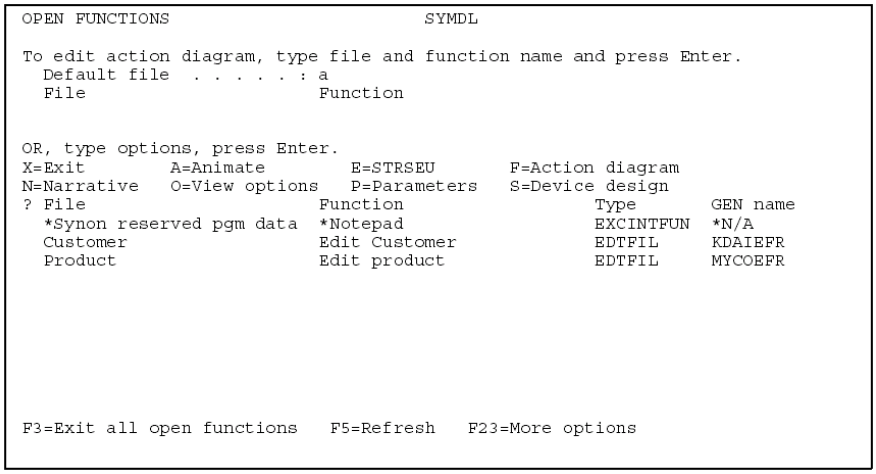
When all the functions you selected are loaded, the Open Functions panel appears and you can begin editing. You do not need to wait while the next function you want to edit is loaded.

Starting an Edit for Another Function

To edit other functions while in the action diagram of a particular function, execute the following steps:

1. Go to Open Functions. At the Edit Action Diagram panel of the function that you are currently editing, press F15.

   The Open Functions panel appears. From the Open Functions panel you have the ability to perform any edit functions on open functions, including changing parameters, accessing diagrams, editing source, displaying usages and references, editing narrative, and changing the device design.



2. At the Open Functions panel, type the file name and the function name in the File and Function fields. If you are uncertain of the names type **?** in the field prompts. Enter ***** in the File field to default to the first file in the list. Additionally, you can use function implementation names(GEN names) if that is more convenient. To open a function using the implementation name, enter the characters ***** and then **i** or **I** (*I or *i) in the File field and then enter the implementation name in the Function field. Implementation names are not case sensitive.

   If the function is not already on the open function list, CA 2E loads the action diagram of the open function and you can perform any necessary editing.

   If it is already on the list, you are returned to the function. You can press F15 at any time during the edit to view the open functions on the Open Function panel.

   Note: Pressing F15 to display the Open Functions panel disrupts the zoom sequence of any open function. Each zoomed function appears as a separate open function and you do not automatically return to the calling function on exit. You instead, return to the Open Function panel from which you must explicitly reselect the calling function.

3. Once you perform any editing changes, press F3 to exit, and save the function. You return to the Open Functions panel.

   The function whose action diagram you modified and saved no longer appear on the Open Functions panel, as it is no longer open.

   Only those functions you opened and have not exited remain open and appear on the Open Functions panel.

Copying from One Function's Action Diagram to Another Using NOTEPAD

To copy the contents of one action to another, execute the following steps:

1. At the Edit Action Diagram panel, specify the construct or block of constructs that you wish to append to the Notepad by entering the appropriate Notepad line command; NA, NAA, NR**,** or NRR.
2. Press F15 to access the Open Functions panel and type **F** against the function to which you want to copy.
3. Access the appropriate user point in the selected function and press NI to insert the contents of the Notepad.

Modifying Function Parameters

From the Edit Action Diagram panel press F15 to access the Open Functions panel and type **P** against the function whose parameters you wish to modify.

The Edit Function Parameter panel appears.

Switching from Action Diagram Directly to Function Device Design

From the Edit Action Diagram panel, press F15 to access the Open Functions panel and type **S** against the function whose device design you want to access.

The device design for the function appears.

Exiting Options

There are several options for exiting a function in an action diagram. They are as follows:

Exiting a Single Function

At the Edit Action Diagram panel, press F3 to exit a single function. Alternatively, at the Open Functions panel, type **X** against an open function.

The Exit Function Definition panel appears.

```
EXIT FUNCTION DEFINITION              SYMDL

Type choices, press Enter.

  Change/create function. . . . Y                         Y=Yes, N=No

      Function name . . . . . . Edit a                    Name
      Access path name. . . . . Retrieval index           Name
      File name . . . . . . . . a                         Name
      Function type . . . . . . Edit file


  Print function. . . . . . . . N                         Y=Yes, N=No

                                N

  Submit generation . . . . . . N                         Y=Yes, N=No



F5=Refresh    F12=Cancel    F15=Open Functions
```

The default value for the Change/Create Function option depends on the setting of the Default Action Diagram Exit Update (YACTUPD) model value; it does not depend on whether you changed the function unless YACTUPD is set to *CALC. As a result, if you want to save your changes, be sure this value is Y before you press Enter.

If YACTUPD is set to *CALC, the Change/create function option is set to Y only when a change to the function's action diagram or panel design has been detected.

Exiting All Open Functions

At the Open Functions panel, press F3 to exit all open functions.

The Exit Function Definition panel displays for each function on the open function list. This allows you to process each exit individually.

Exiting a Locked Function

If you are working in Open Functions with a function that is locked or open to another user of type ***PGMR**, you can still make changes to the function. However, when you exit the function to save the definitions you can only apply changes to the locked function once it is released.

Interactive Generation or Batch Submission

To generate a function use the following instructions:

1. Submit a function to batch generation. At the Exit Function Definition panel, specify Y on the Submit generation field.

   Alternately, at the Edit Functions panel or at the Display All Functions panel, place a J against the function, or use option 14 from the Edit Model Object List panel.

2. Generate a function interactively. At the Edit Functions panel or the Display All Functions panel, place a G against the function.

For more information on generation and batch submission, see this module, Chapter 10, "Generating and Compiling."

Understanding Action Diagram User Points

Each function type that includes an Action Diagram contains protected control logic and a set of user points unique to the function. CA 2E restricts modification of the function's logic to these user points. It is essential that the user know the consequences of any logic that is specified in the user points since it has a direct bearing on the performance and functionality of the function.

The following information includes examples of the types of logic appropriate for various user points.

Many of the user points are self-explanatory and the user will know intuitively what type of processing should be specified. The following information is intended for illustrative purposes only. This information is meant to serve as a guide to aid the user in deciding where to insert function logic as well as what type of logic to insert to affect a certain type of functionality.

For more information and a flowchart for each CA 2E function showing its basic processing and its user points, see this module, Appendix A, "Function Structure Charts."

Change Object (CHGOBJ)

USER: Processing Before Data Read

USER: Processing if Data Record Not Found

By default, the program context return code field (PGM.*Return code) is set to *DBF record does not exist. If necessary, insert a move at this point to set return code to *Normal.

For processing that requires the creation of a DBF record, if the record based on the input key parameters is not found, you can insert a CRTOBJ function here and use the input values of the fields passed into this function (from the PAR context) as input parameters. This is the preferred method for doing this since it involves considerably fewer I/O resources than using a RTVOBJ to read a record and the executing a CRTOBJ or CHGOBJ based on the result of the read.

USER: Processing After Data Read

At this user point, data has been read from the file but not overlaid by data from incoming parameters. You can use this user point to compare for differences in the before and after images of records. You can then use this comparison to effect updates to the file.

USER: Processing Before Data Update

At this user point, data has been moved from the incoming parameters to the file fields. This is often used to set a date/time stamp in the record.

USER: Processing After Data Update

You can use this user point to perform updates to related files. For example, to increment totals based on the differences computed in the USER: Processing After Data Read user point.

Create Object (CRTOBJ)

USER: Processing Before Data Read

USER: Processing Before Data Update

Insert logic here to increment key values of records that you want to add. For example, to add records to a file with a sequential key: Retrieve the last written key value (for example, order line number) and increment it by one to obtain the key value of the next record to be written.

USER: Processing if Data Record Already Exists

By default, the program context field return code (PGM.*Return code) is set to *DBF record already exists. If necessary, insert a move at this point to set the field value to *Normal.

USER: Processing if Data Update Error

By default the program context field return code (PGM.*Return code) is set to *DBF update error. Insert a move at this point to set the field value to *Normal if necessary.

USER: Processing after Data Update

You can use this user point to update any associated file with cumulative totals, or to automatically create extension records.

Delete Object (DLTOBJ)

USER: Processing Before Data Update

CA 2E performs referential integrity checking on the data input of an application. If you want to delete data, you must perform your own checking in the action diagram. If you want to prevent the deletion of a record with references to it, insert a call here to a RTVOBJ function based on the file to be checked, (that is, any file that refers to this file). Build the RTVOBJ function over a RSQ access path that is keyed by the foreign key fields. If a record is found, set the return code to *User QUIT requested; if a record is not found, set the return code to *Normal. Check the return code; if the return code is *Normal, quit the function and send an error message if necessary.

To perform a cascading deletion of subordinate file records, insert an EXCEXTFUN function containing separate RTVOBJ function calls for each file that potentially contains records to be deleted. For each RTVOBJ, define restrictor parameters based on the higher key order of the super ordinate file. This retrieves all possible records to delete. Insert a DLTOBJ function to delete each record in the USER: Process DBF Record user point of the RTVOBJ function.

USER: Processing Before Data Read

Display File (DSPFIL)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads with this user point. Implement security checking and specify an *EXIT PROGRAM action if the user is not authorized.

The program context scan limit field, PGM.*Scan limit, is set to 500 by default. If you want to change this value, do so here.

USER: Initialize Subfile Control

Initialize subfile control fields from passed parameters that are not mapped or from other database file reads.

USER: Initialize Subfile Record from DBF Record

Insert logic to execute further record selection processing. Set the program context field record selected to no, PGM.*Record selected = *NO for records that do not meet the criteria. This procedural level processing is useful when the majority of the records are to be selected but you do not want to build, or cannot build, the select/omit criteria into the access path. If you want all subfile records to be reprocessed after validation, insert the program context reload subfile field here (PGM.*Reload subfile = Yes).

Note: An action to insert a *QUIT function in this user point inhibits the subfile load but does not properly condition the roll indicator.

Check for hidden fields in the subfile control as well as the operators on the subfile fields (particularly CT (contains) for alphanumeric fields) to ensure that proper records display.

Function fields of type CNT, MAX, MIN, and SUM are not allowed for this function type; however, you may want to keep running totals of subfile fields. To do this, you can add a function field to the subfile control of type USR and calculate it at subfile load time. This function type only loads a single page at a time; therefore, any calculations should be performed at the single record level or using the cumulative totals of the subfile record.

CALC: Subfile Control Function Fields

Calculations associated with a derived function field are inserted in this user point.

USER: Process Subfile Control (Pre-Confirm)

Insert references to function keys (using the CTL.*CMD key) here if you want to execute the function key without regard to the validity of detail screen format. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing function keys, you should place the processing in the USER: Process Command Keys user point.

Based on the results of these calls (such as adding a record), you may want to set the program context reload subfile to yes, (PGM.*Reload subfile = *YES), to refresh the panel with any changed data.

Implement checks of key processing F13 or other function keys that cause the action *EXIT PROGRAM to be executed.

CALC: Subfile Record Function Fields

Calculations associated with a derived function field are inserted by CA 2E in this user point. You can add any other actions at this user point. This user point is executed when subfile record is loaded or initialized and when it is processed as a changed record. This means that this user point is useful for activities that need to be performed at both of these times.

USER: Process Subfile Record (Pre-Confirm)

Insert subfile selections (using the RCD.*SFLSEL key) here if you want to execute the subfile selections without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing subfile selections, you should place the processing in the USER: Process Command Keys user point.

Any validation for other fields on the subfile record should go here.

Same considerations as for the previous USER: Process Subfile Control (Pre-confirm).

USER: Process Subfile Record (Post-Confirm)

This user point is present only if you specify the function option for a post confirm pass. You can use this user point to implement any processing on the subfile record after editing and confirmation steps.

USER: Process Subfile Record (Post-Confirm Pass)

Insert logic here to implement processing for each subfile record that has been modified or flagged for additional processing.

USER: Process Command Keys

This user point executes after all other processing and confirmation steps have completed. You can insert function key processing at this point or perform any operations that are related to the panel as a whole.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic is executed here whenever the F3=Exit function key is pressed or an Exit action is executed.

Display Record (DSPRCD)

Processing for the DSPRCD2 and DSPRCD3 function types is similar to DSPRCD; they differ only in the number of panels processed.

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking and specify an *EXIT PROGRAM action if the user is not authorized.

USER: Load Detail Screen from DBF Record

Initialize detail fields from passed parameter fields that are not mapped or from other database file reads.

USER: Process Key Screen Request

Returns processing to key panel.

CALC: Detail Screen Function Fields

Calculations associated with derived function fields in the detail format appear here.

USER: Validate Detail Screen

Insert references to function keys, using the CTL.*CMD key, here if you want to execute the function key. You should implement these calls before confirmation of panel processing. If you want the data record to be processed and validated prior to executing function keys, you should place the processing in the USER: Process Command Keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing F15 or other function keys that cause the action *EXIT PROGRAM to be executed.

USER: Perform Confirmed Action

Insert any processing you want to occur after you press Enter.

USER: Process Command Keys

This user point is always executed unless exit processing is requested (F3 in *CUAENTRY; an action in *CUATEXT). You can insert an *EXIT PROGRAM action at this point if you want to exit after DBF updates.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM action (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic is executed here whenever the F3=Exit function key is pressed or an Exit action is executed.

Display Transaction (DSPTRN)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking, and specify an *EXIT PROGRAM action if the user is not authorized.

USER: Initialize Subfile Record

Initialize fields in the subfile record format, if necessary.

USER: Validate Header Non-key Fields

Insert references to function keys, using the CTL.*CMD key, here if you want to execute the function key without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing function keys, you should place the processing in the USER: Process Command Keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing; F15 or other function keys that cause the action *EXIT PROGRAM to be executed.

USER: Validate Header Non-key Relations

Implement subfile control field-to-field validation or other processing dependent upon prior logic.

USER: Validate Subfile Record Fields

Insert references to subfile selections, using the RCD.*SFLSEL key or equivalent CUA action here if you want to execute the subfile selections without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing subfile selections, you should place the processing in the USER: Process Command Keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing F15 or other function keys that cause the action *EXIT PROGRAM to execute.

USER: Validate Subfile Record Relations

Implement subfile record field-to-field validation or other processing dependent upon prior logic.

CALC: Subfile Record Function Fields

Calculations associated with derived and subfile function fields (SUM, MIN, MAX, and CNT) in the subfile record appear here. You can add any actions at this user point. This user point is executed when the subfile record is loaded or initialized and when it is processed as a change record. This means that this user point is useful for activities that need to be performed at both of these times.

Note: Although subfile function fields operate on subfile record fields, you must place them in the subfile control format of the device panel.

CALC: Header Function Fields

Calculations associated with derived function fields in the subfile control appear here.

USER: Validate Totals

DSPTRN, like EDTTRN, gives you an extra user point before you press Enter. This allows you to perform useful validations; in particular, relational checks between a DBF field and a function field. An example is, a comparison between a customer's credit limit and his outstanding balance plus an order total in an Order Entry function. You can then set the program context defer confirm field to not confirm, PGM.*Defer confirm = *Do not confirm, if the validation fails and you do not want DBF updates to occur.

USER: Header Update Processing

If you want to perform your own header record DBF updates, you should insert your own DBF function objects here.

USER: Subfile Record Update Processing

If you want to perform your own detail record DBF updates, you should insert your own DBF function objects here.

USER: Process Command Keys

This user point is always executed unless exit processing is requested (F3 in *CUAENTRY; an action in *CUATEXT). You can insert an *EXIT PROGRAM action at this point if you want to exit after DBF updates.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM action (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic is executed here whenever the F3=Exit function key is pressed or an Exit action is executed. To reload the subfile and remain in the function, you can set the program context continue transaction field to no, PGM.*Continue transaction = *NO, and insert a QUIT statement here.

Edit File (EDTFIL)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking and specify an EXIT PROGRAM action if the user is not authorized.

You can set the program context program mode field (PGM.*Program mode) to *ADD or *CHANGE. You can test this value at any time in order to perform conditional processing. If you want to change the mode in the action diagram after the panel displays, you must also set the program context reload subfile field to yes (PGM.*Reload subfile = *YES).

USER: Initialize Subfile Header

Initialize subfile control fields from passed parameters fields that are not mapped or from other database file reads.

USER: Initialize Subfile Record (New Record)

This user point is executed if records do not exist (PGM.*Program mode = *ADD).

USER: Initialize Subfile Record (Existing Record)

This user point is executed if records exist (PGM.*Program mode = *CHANGE).

Insert logic to perform further record selection processing. Set the program context field record selected to no (PGM.*Record selected = *NO) for records that do not meet the criteria. This procedural level processing is useful when the majority of the records are to be selected but you do not want to build or cannot build the select/omit criteria into the access path.

Note: An action to insert a QUIT function in this user point may produce unpredictable results.

Check for hidden fields in the subfile control as well as the operators on the subfile fields (particularly CT [contains] for alphanumeric fields) to ensure that proper records display.

Function fields of type CNT, MAX, MIN, and SUM are not allowed for this function type; however, you may want to keep running totals of subfile fields. To do this, you can add a function field to the subfile control of type USR and calculate it at subfile load time. This function type only loads a single page at a time; therefore, any calculations should be performed at the single record level or by using the cumulative totals of the subfile record.

CALC: Subfile Control Function Fields

Calculations associated with derived function fields are inserted in this user point.

USER: Validate Subfile Control

Insert references to function keys (using the CTL.*CMD key) here if you want to execute the function key without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing function keys, you should place the processing in the USER: Process Command Keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Based on the results of these calls (adding a record, for instance), you may want to set the program context reload subfile to yes (PGM.*Reload subfile = *YES) to refresh the panel with any changed data.

Implement checks of key processing (F15 or other function keys that cause the action *EXIT PROGRAM to be executed).

USER: Validate Subfile Record Fields

Insert subfile selection calls to other functions using function keys that are specified in the record context subfile select field (RCD.*SFLSEL) and not in the USER: Process Command Keys user point described following.

Based upon the results of these calls (for example, zooming to change a record in a subordinate file), you may want to set the program context reload subfile field to yes, PGM.*Reload subfile = *YES, to reflect changes in the subfile records.

CALC: Subfile Record Function Fields

Calculations associated with derived function fields in the subfile record appear here. You can add any other actions at this user point. This user point is executed when the subfile record is loaded or initialized and when it is processed as a change record. This means that this user point is useful for activities that need to be performed at both of these times. You may see use User: Validate Subfile Record Relations user point instead since it has a similar pattern of execution.

USER: Validate Subfile Record Relations

Implement field-to-field validation or other processing that is dependent upon prior logic in this user point. This user point is executed at both the initial subfile load and at changed record processing. If you have no need of repeating changed record processing, include the logic in the Initialize Subfile Record user point instead.

If, for the purposes of validation, you do not want to execute the Update Database user points, you must set on the program context defer confirm field (PGM.*Defer confirm = *DEFER CONFIRM). Processing returns to the top of the loop that processes the panel.

USER: Create Object

CA 2E inserts object creation logic at this point if the function option for object creation is set to yes (Y).

USER: Delete Object

CA 2E inserts object deletion logic at this point if the function option for object deletion is set to yes (Y).

USER: Change Object

CA 2E inserts object modification logic at this point if the function option for object modification is set to yes (Y).

USER: Extra Processing After DBF Update

Place additional action diagram logic is this user point if you have additional files that need to be updated. For instance, if you are adding a record, you may want to include additional Create Object functions for files that are not automatically linked to this file.

USER: Process Command Keys

This user point is always executed unless exit processing is requested (F3 in *CUAENTRY; an action in *CUATEXT). You can insert an *EXIT PROGRAM action at this point if you want to exit after DBF updates.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM action (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic is executed here whenever the F3=Exit function key is pressed, or an Exit action is executed.

Edit Record (EDTRCD)

Processing for the EDTRCD2 and EDTRCD3 function types is similar to EDTRCD. They differ only in the number of panels processed.

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking, and specify an *EXIT PROGRAM action if the user is not authorized.

USER: Initialize Detail Screen (New Record)

This user point is executed if the record does not exist, PGM.*Program mode = *ADD.

USER: Initialize Detail Screen (Existing Record)

This user point is executed if record to maintain exists, PGM.*Program mode = *CHANGE. Initializes detail fields from passed parameter fields that are not mapped, or from other database file reads.

USER: Process Key Screen Request

Returns processing to the key panel.

USER: Delete Object

CA 2E inserts object deletion logic at this point if the function option for object deletion is set to yes (Y).

USER: Validate Detail Screen Fields

Insert references to function keys here using the CTL.*CMD key if you want to execute the function key without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want the data record to be processed and validated prior to executing function keys, place the processing in the USER: Process Command Keys user point.

If you are using some sort of preemptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing (F15 or other function keys that cause the action *EXIT PROGRAM to be executed).

You should avoid calling another function from a panel that updates the record you are maintaining since you are likely to receive the error message, Record Has Been Updated by Another User, when this function attempts a database update. This is because the record image on the database has changed since it was last saved.

CALC: Detail Screen Function Fields

Calculations associated with derived function fields in the detail format appear here.

USER: Validate Detail Screen Relations

Implement field-to-field validation or other processing that is dependent upon prior logic in this user point.

If, for the purposes of validation, you do not want to execute the update database user points, you must set on the program context defer confirm field, PGM.*Defer confirm = *DEFER CONFIRM. Processing returns to the top of the loop, which processes the panel.

USER: Create Object

CA 2E inserts object creation logic at this point if the function option for object creation is set to yes (Y).

USER: Change Object

CA 2E inserts object modification logic at this point if the function option for object modification is set to yes (Y).

USER: Process Command Keys

This user point is always executed unless exit processing is requested (F3 in *CUAENTRY; an action in *CUATEXT). You can insert an *EXIT PROGRAM action at this point if you want to exit after DBF updates.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM action (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic is executed here whenever the F3=Exit function key is pressed, or an Exit action is executed.

Edit Transaction (EDTTRN)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking, and specify an *EXIT PROGRAM action if the user is not authorized.

USER: Initialize Screen for New Transaction

This user point is executed if the header record does not exist, PGM.*Program mode = *ADD. Initialize fields in the control format, if necessary.

USER: Initialize Screen for Old Transaction

This user point is executed if the header record exists PGM.*Program mode = *CHANGE. Initialize fields in the control format, if necessary.

USER: Validate Header Key Fields

Executed if the program context field program mode field is Add, PGM.*Program mode = *ADD.

USER: Validate Header Key Relations

Executed if the program context program mode field is Change, PGM.*Program mode = *CHANGE.

USER: Load Existing Header

The existing header record format is loaded into the subfile control format.

USER: Initialize Subfile Record (Old Record)

Initialize fields in the subfile record format, if necessary. This user point is executed if the program context program field is Change PGM.*Program mode = *CHANGE.

USER: Initialize Subfile Record (New Record)

Initialize fields in the subfile record format, if necessary. This user point is executed if the program context program mode field is Add, PGM.*Program mode = *ADD.

USER: Validate Header Non-key Fields

Insert references to function keys (using the CTL.*CMD key) here if you want to execute the function key without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing function keys, you should place the processing in the USER: Process command keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing (F15 or other function keys that cause the action *EXIT PROGRAM to be executed).

USER: Validate Header Non-key Relations

Implement subfile control field-to-field validation or other processing dependent upon prior processing logic.

USER: Validate Subfile Record Fields

Insert references to subfile selections here (using the RCD.*SFLSEL key or equivalent CUA action) if you want to execute the subfile selections without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing subfile selections, you should place the processing in the USER: Process command keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of key processing (F15 or other function keys that cause the action *EXIT PROGRAM to be executed).

USER: Validate Subfile Record Relations

Implement subfile record field-to-field validation or other processing dependent upon prior logic.

CALC: Subfile Record Function Fields

Calculations associated with derived and subfile function fields (SUM, MIN, MAX, and CNT) in (or which operate on) the subfile record appear here. You can add any other actions at this user point. This user point is executed when the subfile record is loaded or initialized and when it is processed as a new change record. This means that this user point is useful for activities that need to be performed at both of these times.

Note: Although subfile function fields operate on subfile record fields, you must place them in the subfile control format of the device panel.

CALC: Header Function Fields

Calculations associated with derived function fields in the subfile control appear here.

USER: Validate Totals

EDTTRN provides you with an extra user point before the DBF update user points. This user point allows you to perform useful validations, in particular, relational checks between a DBF field and a function field. For example, a comparison between a customer's credit limit and his outstanding balance plus an order total in an Order Entry function. You can then set the program context defer confirm field to do not confirm (PGM.*Defer confirm = *Do not confirm) if the validation fails and you do not want DBF updates to occur.

USER: Create Header DBF Record

CA 2E inserts default object creation logic if the transaction creation function option is set to yes (Y).

USER: Change Header DBF Record

CA 2E inserts default object modification logic if the change transaction function option is set to yes (Y). You may substitute for this, which would be useful in the event where the header format is output only (determined by you) and you want to suppress the header record. Replace the Change object function at this point with a dummy internal function that performs no essential function.

USER: Delete Header DBF Record

CA 2E inserts default object deletion logic if the transaction deletion option is set to yes (Y).

USER: Create Detail DBF Record

CA 2E inserts default object creation logic if the detail line creation function option is set to yes (Y).

USER: Change Detail DBF Record

CA 2E inserts default object modification logic if the change transaction function option is set to yes (Y).

Note: EDTTRN updates all records in the subfile, whether you have changed them or not.

USER: Delete Detail DBF Record

CA 2E inserts default object deletion logic if the detail line object deletion function option is set to yes (Y).

USER: Process Detail Record

If you have substituted dummy functions for any of the DBF updates, you want to insert your own DBF objects at this point.

USER: Process Command Keys

This user point is always executed unless exit processing is requested (F3 in *CUAENTRY; an action in *CUATEXT). You can insert an *EXIT PROGRAM action at this point if you want to exit after DBF updates.

USER: Exit Program Processing

Insert function key processing with a user-specific return code to execute an *EXIT PROGRAM action (F3 in *CUAENTRY; an action in *CUATEXT). The *EXIT PROGRAM logic will be executed here whenever the F3=Exit function key is pressed, or an Exit action is executed. To reload the subfile and remain in the function, you can set the program context continue transaction field to no (PGM.*Continue transaction = *NO) and insert a QUIT statement here.

Print File (PRTFIL) – Print Object (PRTOBJ)

USER: Initialize Program

Initialize work fields from passed parameter fields, constants or other database file reads.

USER: Record Selection Processing

Implement further logic to restrict records that are to be printed. Set the program context record selected field to no (PGM.*Record selected = *NO) to prohibit records from being printed. This should not be done as the primary means of record selection (use select/omit criteria on the access path instead) but rather to filter out based on some functional criteria, a small percentage of the records that you want to exclude from the access path.

USER: Process Top of Page

This format is only available for PRTFIL.

USER: Null Report Processing

This user point is executed if no records exist to print on the report.

USER: On Print of File nnn Key xxx Format

For each field in the key of the access path over which the function is built there is a format to print the required level headings for the key field (control break). Each format has user points to total and format fields before, during, or after the format print.

Note: Print object calls are placed immediately before or after the On Print user point.

USER: On Print of Detail Format

CA 2E formats fields from the DB1 file context into the Current (CUR) context of the format.

USER: On Print of End of Report Format

This format is only available for PRTFIL.

Prompt and Validate Record (PMTRCD)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking and specify an *EXIT PROGRAM action if the user is not authorized.

USER: Load Screen

Format the detail panel from parameters, shipped file (PGM or JOB) values, or reads to other database files.

USER: Process Command Keys

Insert calls to other functions using the functions keys that are specified in the control context function key field (CTL.*CMD key).

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

Implement checks of pre-emptive key processing in a user point (F15 or other function keys in *CUAENTRY; an action in *CUATEXT) that execute an *EXIT PROGRAM action.

USER: Validate Fields

If you use PMTRCD as a sub-menu, you normally have a function field of type USR on the panel that allows you to enter valid options. You may also have other USR type fields to process information that is validated against the database. Implement any validation, including existence checking, on these fields in this user point.

CALC: Screen Function Fields

Calculations associated with a derived function field are inserted in the subfile record at this point.

USER: Validate Relations

Check for field dependencies: the value of one field is conditioned on the value of another field(s); for example, a range where the first number must be less than or equal to the second.

USER: User Defined Action

If your PMTRCD is a sub-menu, insert calls to appropriate functions based on user-entered data at this point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check relevant return codes facilitating this processing upon return from any called functions.

USER: Exit Program Processing

Place pre-emptive key processing requiring exit processing (F3 in *CUAENTRY; an action in *CUATEXT) to execute an *EXIT PROGRAM action with a user-specific return code since this user point is always executed when exit is requested.

Retrieve Object (RTVOBJ)

USER: Initialize Routine

Unless reading a single record for existence checking or retrieving values, much of your processing is specified from within the calling function itself. In this user point, you should initialize work fields, counters, and calculation fields.

USER: Processing if Data Record Not Found

The program context return code field is set to *Record Does Not Exist. In many instances this is the default processing; however, you should set the return code field to *Normal. Do this with a move statement.

It is good practice to insert a *MOVE ALL built-in function specifying (CON.*BLANKS) if any data is retrieved by the RTVOBJ function.

Note: If user logic exists in this user point and the record is not found, then user logic in USER: Exit processing is ignored.

USER: Process Data Record

When reading for a single record with a partially restricted key, you must insert a *QUIT statement in this user point when you have found the requested record. If you are performing an existence check, you should insert a *QUIT statement once you have found the record since you do not want to read the entire file. DB1 fields must be moved to the PAR context to return field values to the calling function. If your fields match, you can use the *MOVE All statement to execute this. You must explicitly format other fields with a *MOVE statement. The parameters for which you want to return field values must be specified as O (Output) or B (Both) parameters.

When reading with a fully restricted key, if the record is found and there is no user logic in this user point, processing stops. You must have user logic in this user point if you want to read more than one record.

RTVOBJ is often used to direct a batch process. Insert any functions in this user point that are required to implement your processing: EXCEXTFUN, EXCINTFUN, CHGOBJ, CRTOBJ, and other RTVOBJ.

USER: Exit Processing

You may have defined work fields that store data values while the next record is being processed. This user point is executed when processing has completed. Use the DB1 context carefully in this user point since you have not read another record and unpredictable results may occur.

Note: User logic in this user point is not executed if user logic exists in user point USER: Processing if Data Record Not Found and no record is found.

Select Record (SELRCD)

USER: Initialize Program

Initialize work fields from passed parameters or from other database file reads. Implement security checking and specify an *EXIT PROGRAM action if the user is not authorized.

The program context field *Scan Limit (PGM.*Scan Limit), which is used for establishing the number of records to read, is set to 500 by default. If you want to change this value, do so here.

USER: Load Subfile Record from DBF Record

You can insert function field processing at this point, such as field and file descriptions that are accessed using a RTVOBJ to another file.

USER: Process Subfile Control

Insert references to function keys (using the CTL.*CMD key) here if you want to execute the function key without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing function keys, you should place the processing in the USER: Process command keys user point.

If you are using some sort of pre-emptive function control or fast path processing, be sure to check the relevant return codes facilitating this processing upon return from any called functions.

Based on the results of these calls (adding a record, for instance), you may want to set the program context reload subfile field to yes (PGM.*Reload subfile = *YES) to refresh the panel with any changed data.

Implement checks of key processing (F15 or other function keys that cause the *EXIT PROGRAM action to be executed).

USER: Process Selected Line

The function exits at this point if you have selected a subfile record.

USER: Process Changed Subfile Record

If relevant, insert subfile selections (using the RCD.*SFLSEL key) here if you want to execute the subfile selections without regard to the validity of subfile records. You should implement these calls before confirmation of panel processing. If you want subfile data records to be processed and validated prior to executing subfile selections, you should place the processing in the USER: Process Command Keys user point.

Similar considerations as for DSPFIL.

CALC: Screen Function Fields

Calculations associated with a derived function field are inserted in this user point.

USER: Process Command Keys

This user point is always executed unless the function key or action associated with the *Exit field is requested.

USER: Exit Program Processing

Place pre-emptive key processing requiring exit processing (F3 in *CUAENTRY; an action in *CUATEXT) to execute an *EXIT PROGRAM action with a user-specific return code here since this user point is always executed whenever exit processing is requested.

Understanding Function Structure Charts

The CA 2E function structure charts provide you with a visual orientation of the user points with respect to other processes, and will help you learn the processing offered by each function type. The function charts appear in Appendix A at the end of this module.

Work your way through each chart as necessary. Evaluate each user point with respect to the rest of the function until you locate the correct point at which you want to introduce your processing logic.

For more information and a diagram of each function structure chart, see this module, in the appendix, "Function Structure Charts."

Chapter 11: Copying Functions

This chapter explains how to copy existing functions to create new ones. In general, CA 2E allows you to copy a function to another function of the same type. In addition, a copy with a change of function type is permitted between certain combinations of function types. This is called cross-type copying. You can also create new functions by copying customized template functions.

As an alternative to the process discussed in this chapter, you can create versions of functions and messages. Some benefits of using versions are:

- You can test changes on a version of a function or message without interfering with the functionality of the existing model.
- When you finish testing a new version of a function or message and make it active in the model, the original model object remains unchanged and can easily be made active again if needed.
- Only the currently active version of a function or message displays on CA 2E editing panels. As a result, the panels are not cluttered with inactive versions.

For more information about versions, see Working with Versions of Functions and Messages, in *Generating and Implementing Applications,* in the chapter, "Managing Model Objects."

This section contains the following topics:

Creating a New Function from One That Exists

You can create a new function from an existing one in the following ways:

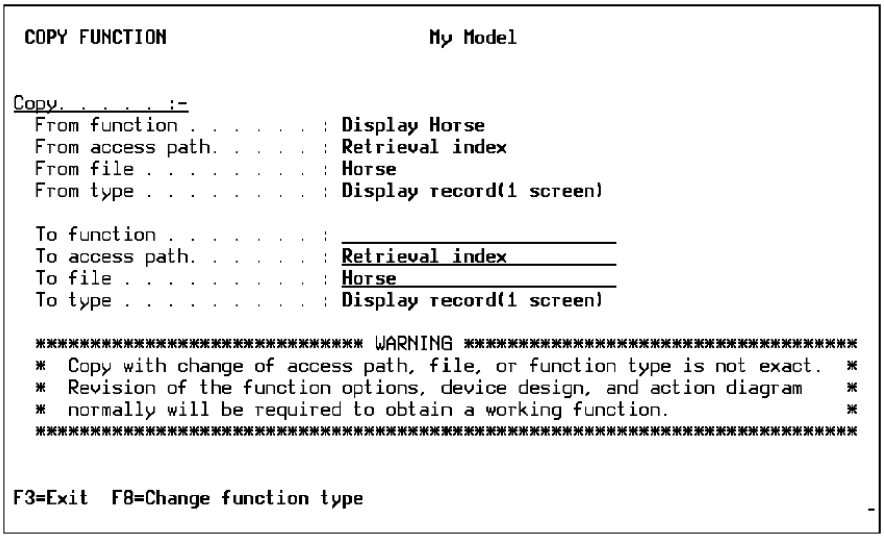
- From the Edit Function panel
- From a template function
- From the Exit panel of the Action Diagram Editor

From the Edit Functions Panel

To copy a function from the Edit Functions panel:

1. Type **C** next to the function you want to copy and press Enter. The Copy Function panel appears.
2. Specify the new function by typing the function, file, and access path names of your new function. If you want to change the function type, use F8.
3. Press Enter to copy the function.

Note: Because copying between files and access paths is not exact, you should revisit the action diagram and device design of the copied function.



From a Template Function

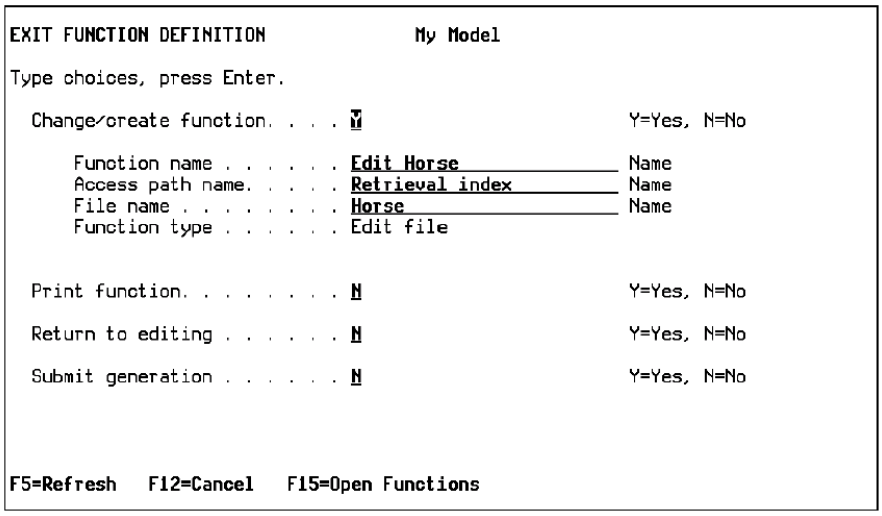
A template function is a customized function that you can copy to create a new function. All internally-referenced functions are automatically mapped to the target file if they are based on the internal *Template file. In other words, the copy facility creates new referenced functions based on the target file if they do not already exist, selecting or creating needed access paths. The result is a set of functions that are as close as possible to a completed version of the functions as if they had been hand-coded.

For more information on template functions, see Function Templates later in this.

From the Exit Panel

To copy a function from the Exit panel of the action diagram editor:

1. Exit the Edit Action Diagram panel of the function you want to copy. The Exit Function Definition panel appears.



2. Specify the function you want to create. Type **Y** in the Change/Create field. You can change any underlined field, including Function Name, Access Path Name, and File Name.
3. Press Enter. This creates your new function.

Cross-Type Copying

For certain function types, you can copy a function of one type to another type of the same style. To change to another type, press F8 at the Copy Functions panel. When you press F8, CA 2E automatically changes the function type in the To Type field.

The functions available for crosstype copying include

From/To	To/From
DSPRCD1	DSPRCD2
DSPRCD2	DSPRCD3
EXCEXTFUN	EXCINTFUN
EDTFIL	DSPFIL
EDTTRN	DSPTRN

From/To	To/From
EDTRCD	DSPRCD
EDTRCD2	DSPRCD2
EDTRCD3	DSPRCD3
EDTRCD1	EDTRCD2
EDTRCD2	EDTRCD3
PMTRCD	DSPRCD1
PMTRCD	EDTRCD1
PRTFIL	PRTOBJ
PRTFIL	RTVOBJ

What Copying Preserves

In copying functions, CA 2E preserves the

- Action diagram (where the user points match)
- Function options

Note: When you copy EXCUSRSRC and EXCUSRPGM functions, only the CA 2E model information is copied. The user portion of code is not copied and the HLL type defaults to the current setting of the HLL to Generate (YHLLGEN) model value.

Output/Input Fields

When you cross-type copy the display/edit functions listed previously, CA 2E changes the output or input capability of fields as follows:

- Edit to display type function all fields are output capable, except key and positioner fields
- Display to edit type function all fields are input capable, except the key field

What to Revisit

Cross-type function copying results in a message to tell you that device design and action diagram changes may be required. Review function options, particularly when copying from display to edit type functions.

Device Design

For device functions, the copy process defaults to the device design for the function type. You can edit the default device design for the newly created function, as appropriate to your requirements. For example, you may want to make specific function keys available to the function.

To change the function keys from the device design:

1. Place your cursor on the function key and press Enter. The Edit Command Text panel appears.
2. Press F5 (Refresh). CA 2E refreshes the function text data with the correct default function key data from the action diagram.

Note: Depending on the type of copy, the device design may not be preserved.

Action Diagram User Points

The action diagram for the new function may refer to fields that do not exist in the file to which the function was copied or there may be invalid context references. You can use the Find Error option from the Action Diagram Services panel to locate such errors. You can then correct the errors in the action diagram.

Note: You can use the Notepad facility of the Action Diagram Editor to copy sections of action diagrams between functions.

After revisiting the action diagram and the device design, generate the function.

Function Templates

A function template usually contains customized actions that you want new functions in your model to contain. Two suggestions for using function templates are:

- Create a *work wit'* suite of functions for maintaining reference (REF) files in your model
- Establish and enforce standards for your organization or department that can automatically be applied when a new function is created

Understanding Function Templates

Any standard function based on the *Template file is known as a template function. A template function can be a primary function or a function internally referenced by a primary function.

When you copy a template function, CA 2E uses the source function as a template to create a new function based on a target file you specify. Function names and access paths are automatically adjusted for the target file.

There is no limit to the complexity of the suite of functions copied. The primary function must be based on the *Template file, but internally-referenced functions can be based on the *Template file or they can be based on normal user-defined files.

Run-time messages list new objects created and indicate where user intervention may be required in the second level text.

Two action diagram features aid the creation of template functions:

- The PR (protected structure) selection option lets you protect blocks that comprise standard areas of the action diagram you do not want developers to remove or change
- You can specify *T for Function file on the Edit Action - Function Name window to select from among existing template functions.

Note: Function templates facilitate the process of creating new functions or suites of functions. There is no inheritance; as a result, changing the template has no effect on functions that were previously created from the template.

Creating a Template Function

This process applies both to the primary template function and to any functions the primary function references that also to serve as template functions.

1. There are two ways to create a template function:
   - Go to the Edit Functions panel for the *Template file and create a new function.
   - Go to the Edit Functions panel for a model file and copy an existing function to the *Template file. This is best if you already have customized functions that can serve as templates.
2. Adjust or create the function as you would any other function:
   a. Name the function. When naming a template function you can place **Template* in the name where you want the target file name to appear; for example, Work with *Template data translates to Work with Customer data for the Customer file.
   b. If the function requires an access path, specify an appropriate one based on the *Template file.
   c. Add standardized actions to the action diagram.
   d. Specify parameters for internally-referenced template functions using the following two fields defined for the *Template file:
   - *Template key defn—If this field is a parameter on a referenced function, it is replaced by the target file's key fields.
   - *Template record defn—If this field is a parameter on a referenced function, it is replaced by the target file's non-key fields.

   Note: In both cases all fields are used.
3. Save the template function.

Special Considerations for EDTTRN/DSPTRN Template Functions

1. Create a span (SPN) access path over the *Template file without formats.
2. Create the EDTTRN or DSPTRN function over the *Template file and specify the span access path you just created.
3. When you copy the EDTTRN/DSPTRN template function to a target file to create a new function, the copy process selects an existing span access path. If a span access path does not exist over the target file, a new span access path is created without formats. You need to add the formats manually.

Using a Template Function to Create a New Function

The following is an outline of one way to create new functions based on a template function.

1. Go to the Edit Functions panel for the file on which the new functions are to be based.
2. Press **F21** (Copy a *Template function). The Edit Functions panel displays all functions based on the *Template file.
3. Type **X** to select the function to be used as the template and press Enter.
4. The Copy Function panel appears.

Another way to create a new function from a template function is to go to the Edit Functions panel for the *Template file and type **C** next to the template function you want to use.

Note: For the primary function, the validation performed by the Copy Function panel for copies from the *Template file is identical to that performed for an ordinary file. Specifically, the access path must exist and the new function must not exist.

You are responsible for verifying and completing the definition of the newly-created target functions.

Copying Internally-Referenced Template Functions

This section describes how the enhanced copy facility processes functions that are called from within the primary (top-level) function. For each called function, the copy process automatically names new functions, selects or creates access paths, and defaults key and non-key parameters.

This table summarizes the process of copying internally-referenced functions.

Source Function based on:	Target Function based on:	Result
*Template file	Model file	If a matching function based on the target file exists, it is used. Otherwise, a new function based on the target file is created. For each new function, the copy process names the function, selects or creates an access path over the target file, and defaults key and non-key parameters.
*Template file	*Template file	Normal copy. Use this to update your template functions.

Source Function based on:	Target Function based on:	Result
Model file	*Template file	Normal copy. This is a way to set up your first template functions if you already have functions containing customized actions.
Model file	Model file	Normal copy. Any parameter requirements of these functions are accommodated by the copy process, by the parameter defaulting mechanism, or require developer intervention.

Creating and Naming Referenced Functions

The copy process first searches the target file for a function with a name matching that of the template function. Note that if the source function name contains **Template*, it is replaced with the name of the target file before the search for matching names; for example, 'Change *Template' translates to 'Change Customer' for the Customer file.

Note: If the DBF functions (CHGOBJ, CRTOBJ and DLTOBJ) attached to the *Template file contain user-defined processing, you need to change their default names; otherwise, the default functions on the target file are used. This is true even if the default DBF functions are not yet been created for the target file.

- If a matching function is found, it is used if its function type is compatible based on the following table and if the function is current and not archived.

Template Function Type	Compatible with These Target Function Types
EXCINTFUN EXCEXTFUN EXCUSRPGM	All except PRTOBJ
CHGOBJ CRTOBJ DLTOBJ	EXCINTFUN / EXCEXTFUN / EXCUSRPGM DSPRCDn / EDTRCDn / PMTRCD
RTVOBJ	EXCINTFUN / EXCEXTFUN / EXCUSRPGM

Template Function Type	Compatible with These Target Function Types
DSPRCDn	EXCINTFUN / EXCEXTFUN / EXCUSRPGM
EDTRCDn	DSPRCDn / EDTRCDn / PMTRCD
PMTRCD	CHGOBJ / CRTOBJ / DLTOBJ
PRTFIL	EXCINTFUN / EXCEXTFUN / EXCUSRPGM
PRTOBJ	Not compatible with any other type

Notes:

1. EXCINTFUN, EXCEXTFUN, and EXCUSRPGM are compatible with all other types (except PRTOBJ, which is incompatible with all other types.
2. The internal functions (except RTVOBJ and PRTOBJ) are compatible with the single record display functions.
3. The subfile function types are fully compatible with each other.

If the type is not compatible, a new function of the same type as the template function is created over the target file.

- A new function is created if the target function:
  - Does not exist
  - Is not current
  - Is an archived object
  - Is an incompatible function type

  The copy process automatically assigns names to new functions. If necessary the surrogate number of the new object is attached to the function name to make it unique.

  Note: It is important to set the names of *Template functions and avoid changing them because subsequent copies to the same target file create new functions if the original functions cannot be found by name.

Assigning Access Paths for Referenced Functions

When the copy process is unable to match an existing function based on the target file, it creates a new function. During this process it often needs to select an appropriate access path also based on the target file. It does this as follows:

1. It tries to find and use an access path of the same name and type (RTV, RSQ, UPD, and so on).
2. If there is no match by name, it selects the default access path of the same type.
3. If a default access path of the same type does not exist, it selects the first access path of the same type alphabetically by name.
4. If unable to find an access path of the same type, it creates a new one. A message displays when a new access path is created.

Note: Since new access paths default to the primary key, you may need to edit new access paths prior to source generation.

If a new span (SPN) access path is created, it is created without formats.

For more information about:

- EDTTRN/DSPTRN functions, see Special Considerations for EDTTRN/DSPTRN Template Functions
- Adding formats to a span access path, see *Building Access Paths*, in the chapter, "Adding Access Paths."

Defaulting Parameters for Referenced Functions

The copy process defaults parameters for internally-referenced functions that are based on the *Template file using the two fields defined for the *Template file:

- *Template key defn—If this field is a parameter on the referenced function, it is replaced with the target file's key fields.
- *Template record defn—If this field is a parameter on the referenced function, it is replaced with the target file's non-key fields.

Note: In both cases all fields are used. As a result, for the *Template key defn field, if this is used as a RST parameter for a template function, the new function is a fully restricted function based on the target file with each key field specified as a restrictor.

Device Designs

You need to edit all device design functions to complete the design.

Chapter 12: Deleting Functions

This chapter explains how to delete a function, which includes removing references to it from other objects in the design model.

This section contains the following topics:

Deleting a Function

This topic includes the steps for finding where a function is used and removing all references to it. If the function you want to delete is not referenced by another function, go directly to the last step.

To delete a function

1. Find where the function is used. To make the inquiry, starting from the Edit Database Relations panel:

   a. Go to the function for the file. From the Edit Database Relations panel, type F (next) to the specific file, and press Enter.

      The Edit Functions panel displays, listing the functions for that file.

   b. Select the function for which you want to find references. Type U **(**next) to the specific function and press Enter.

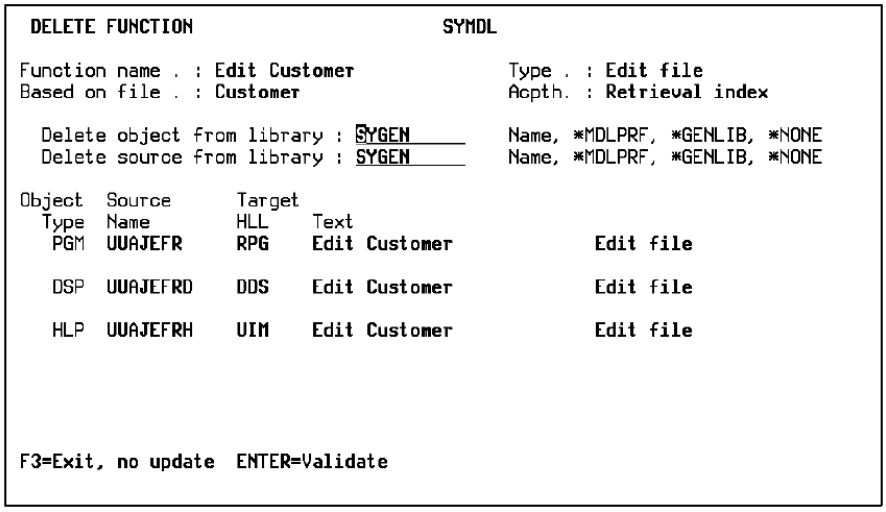
      The Display Function References panel appears, listing all the functions that call the function.

      Note: You can also reach the Display Function References panel from the Display Access Path Functions panel and the Display All Functions panel. You can also determine which model objects reference the function using the Edit Model Object List panel or the Display Model Usages (YDSPMDLUSG) command.

2. Remove the references from the action diagrams. Go into the action diagrams of the functions that call the function you want to delete and remove the logic that calls the function.

3. Delete the function. From the Edit Functions panel, type **D** next to the function you want to delete and press Enter.

   If the function has associated source, a confirm prompt gives you the option of deleting it along with the function once you press Enter.



4. Press Enter again at the confirm prompt. The function and associated source, if any, is deleted.

For more information on editing functions, see Editing and Maintaining Several Functions Simultaneously in the chapter "Modifying Action Diagrams."

Chapter 13: Generating and Compiling

This chapter tells you how to submit a request to generate and compile a function from any one of various CA 2E panels. Complete details on the generation process are contained in *Generating and Implementing Applications*.

This section contains the following topics:

Requesting Generation and Compilation

This topic takes you step by step through requesting generation/compilation from various CA 2E panels. Once you initiate the request, you can submit it from the Display Services Menu.

You can request generation/compilation of functions from one of four panels. They are:

- Display Services Menu (one or more functions)
- Edit Functions panel (one or more functions)
- Exit Function Definition panel (one function at a time)
- Edit Model Object List panel (one or more functions)

The Display Services Menu

To request function generation from the Display Services Menu:

1. Select the Display all functions option. The Display All Functions panel appears.
2. Select the functions you want to generate. Type **J** next to each function you want to generate, and press Enter.
3. Exit. Press F3. The Display Services Menu appears.
4. Submit generations and compilations of all the source members you selected. On the Display Services Menu either:
   - Select the Submit model create request (YSBMMDLCRT) option. Press Enter to display the source members you selected or press F4 to change parameter defaults before displaying the list.
   - Select the Job list menu option to display the Job List Commands Menu. Select the YSBMMDLCRT option.

   A job list of the source members you requested for generation and compilation appears on Submit Model Generations and Creates panel.
5. Review the list before confirming. Press Enter. If the list includes functions you do not want, you can drop (D) or hold (H) them.
6. After you press Enter, the panel redisplays with the confirm prompt set to Y for Yes. Press Enter to confirm the list. CA 2E then submits the generation and compilation jobs.
7. As CA 2E processes the jobs, progress messages appear at the bottom of the panel. Press F5 to refresh the panel for the most current status. Or you can press F3 to exit to the Display Services Menu.

The Edit Functions Panel

The Edit Functions Panel displays when you enter F next to a file on the Edit Database Relations panel. To request function generation from the Edit Functions panel:

1. Request generation. Type J next to the specific functions and press Enter.
2. Go to the Display Services Menu. Press F17, which takes you to the Display Services Menu.
3. Submit the generation request as detailed earlier in the From the Display Services Menu topic, steps 4-5.

The Exit Function Definition Panel

The Exit Function Definition panel displays when you exit the Edit Action Diagram panel. To request generation from the Exit Function Definition panel:

1. Initiate a generation request. In the Submit Generation field, type **Y** (Yes).

   Y in this field is equivalent to entering J next to the function from the Edit Functions panel.

2. Press Enter.

   The Edit Functions panel appears, with the message, "Source generation request for (*name of object*) accepted." The object name is a name such as UUAEEFR.

3. Go to the Display Services Menu. Press F17. This takes you to the Display Services Menu.

4. Submit the generation request as previously described in the From the Display Services Menu topic, Steps 4-5.

The Edit Model Object List Panel

You access the Edit Model Object List panel by entering YEDTMDLLST at a command line.

1. Select the model object list containing the functions you want to generate. For example, enter the model object list name for the List option, or enter ? or *S to display a list of all model object lists in your model. You can press F17 to display the Subset Model Objects panel and request that only functions display.

2. Request generation. Enter selection option 14 for each model object you want generated. This invokes the Create Job List Entry (YCRTJOBLE) command to add the select model objects to the job list. You can specify parameters on the command line. Press Enter**.**

3. Press F19 to display a menu of job-list-related commands. Enter 1 to invoke the Submit Model Create (YSBMMDLCRT) command. See the previous From the Display Services Menu description, Steps 4-5.

For more information about:

- The Model Object List panel, see Edit Model Objects, in the chapter "Managing Model Objects" in the *Generating and Implementing Applications* guide.
- Working with submitted jobs, see Working from the Display Services Menu, in the chapter "Generating and Compiling Your Applications" in the *Generating and Implementing Applications* guide.

Compile Preprocessor

The compile preprocessor is a program that can be automatically invoked to run as a preliminary step on batch compiles.

Chapter 14: Documenting Functions

This chapter explains how to document a function. The Document Model Functions (YDOCMDLFUN) command allows you to print a detailed list of the functions within a model. You can invoke the command from the Display Services Menu or call it from the i OS command line. How you set YDOCMDLFUN parameters determines the level of detail on your listing, including whether narrative text is included.

This section contains the following topics:

Printing a Listing of Your Functions

To print a listing of your functions starting from the Display Services Menu:

1. Access the Display Services Menu. At the Edit Database Relations panel, press F17. The Display Services Menu appears.

   Note: You can also access the Display Services Menu by entering the following at a command line.

   YEDTMDL ENTRY(*SERVICES)

1. Go to the Display Documentation Menu. Select the option, Display documentation menu. The Display Documentation Menu appears.
2. Select functions. Type **5**, Document model functions, and press Enter. The Document Model Functions (YDOCMDLFUN) command panel appears.
3. Set the specific criteria, and press Enter. On this panel you can specify the types of functions and whether you want to list details as function options, parameters, and device designs. CA 2E creates a print file containing the listing.

For more information on using the YDOCMDLFUN command, see the *Command Reference Guide*.

Including Narrative Text

Use the parameter PRTTEXT on the YDOCMDLFUN command to include functional or operational text. Up to ten pages of narrative text can be associated with each CA 2E object. The narrative text can include:

- Functional text, to explain the purpose of the design object
- Operational text, to explain the function of an object for the end user

Note: In generating help panels, CA 2E uses operational text. If no operational text exists, CA 2E uses the functional text.

Comparing Two Functions

The Compare Model Objects (YCMPMDLOBJ) command compares the action diagrams of two functions. This lets you identify any changes made to one version of a function for retrofitting to another version. You can request a printed report of any mismatches encountered. You can also use this command to compare two message functions or two files.

For more information on the Compare Model Objects (YCMPMDLOBJ) command, see *Command Reference Guide*.

Chapter 15: Tailoring for Performance

This chapter provides guidelines for improving the iSeries performance of applications that CA 2E generates. Two major aspects covered here are program size and links between programs.

You can also use the separate CA 2E Performance Expert (PE) option to help you predict how an application will perform. PE is a CA 2E-generated application intended for CA 2E development managers and developers.

For more information on PE, see the *Performance Expert User Guide*.

This section contains the following topics:

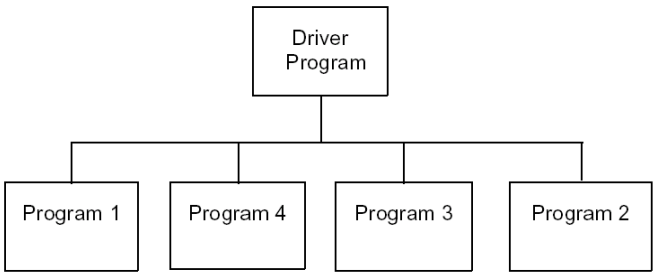
Building an Application

There are two approaches to the structure in building an application:

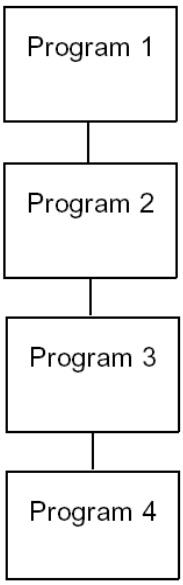
- Vertical, in which each program calls another at a lower invocation level. Avoid using this structure, as it is inefficient.
- Horizontal, in which a driver program calls whichever program is needed. This structure affords more control of the programs.

The following example compares both structures.

Horizontal structure



Vertical structure



Using Display File, not Menu Options

On systems where end users are likely to work with the same objects for long periods, consider using a Display File (DSPFIL) function as the driver program. This program displays existing objects and prompts the end user for action through subfile selection (except add which is F9). Using menu options, the application has to open and close files frequently, which slows performance. A DSPFIL provides a better performing solution.

Note: You can also use an Execute External Function (EXCEXTFUN), Execute User Program (EXCUSRPGM), or Prompt Record (PMTRCD); whichever is appropriate, as the driver program.

For more information on functions, see Function Types, Message Types, and Function Fields, in the chapter "Defining Functions."

Determining Program Size

Determining the right size for programs is relative to your business needs. Some applications benefit from large, complex programs to simplify navigation for the user. However, keeping programs small has several advantages including:

- Reusable components (code dedicated to the data it processes)
- Easier maintenance
- Simpler debugging
- Quicker generation and compilation
- Less code duplication
- More flexibility for grouping processes

Large programs are prone to dead code, used only once or not at all. Breaking processes into smaller programs allows you to identify such areas of code. You can selectively invoke them or remove them following completion.

Your panel design requirements should determine how you create the main function. However, within a given transaction from this program, several functions can be executed. You can make some of these functions the function type, EXCEXTFUN, to encapsulate functions or to isolate seldom executed functions.

Optimizing Program Objects

Optimizing program objects can significantly improve performance. Use the i OS commands, Create COBOL or RPG Program (CRTCBLPGM or CRTRPGPGM) and Change Program (CHGPGM), as follows:

```
CRTxxxPGM PGM(library-name/program-name) +
   SRCFILE (library-name/source-file-name) +
   GENOPT(*OPTIMIZE)
```

Note: This is done by altering the parameters on the command in the *Messages file.

```
CHGPGM PGM(library-name/program-name) +
   OPTIMIZE(*YES)
```

For more information on the create commands and optimization parameters, see *Application System/400 Programming: Control Language Reference*.

Fine Tuning

In tuning the performance of your application, consider these recommendations:

- Restrict the use of subfile control selectors to essential fields, especially on large files. Drop those that are not required. This applies to SELRCD, DSPFIL, and EDTFIL functions.
- Minimize the use of virtual fields. That is, use access paths with the least virtuals possible, since using them involves more processing. Where possible, direct processing to read one file instead of join logicals over multiple files. If appropriate, use the Retrieve Object (RTVOBJ) function instead.
- Consider the amount of Refers to referential checking:
  - Drop unused relations using the Edit Format Relations panel in the device design editor
  - Set relations to user checking where a field is required but referential checking is not
- Drop fields from panel formats if the application does not need them, rather than hiding the fields.
- If you need to validate many files, consider the use of Share Open Data Path when an access path is used frequently by several successive programs with fully restricted key access.
- Create native objects; that is, model value YCRTENV is set to QCMD (iSeries creation environment).

There are also closely related aspects to tailoring access paths.

For more information on tailoring access paths, see Building Access Paths in the chapter "Tailoring for Performance."

Selecting the Function Type

Edit Transaction (EDTTRN) and Display Transaction (DSPTRN) function types load the entire subfile within the limits of any restrictor parameters, if any. On the other hand, Edit File (EDTFIL) and Display File (DSPFIL) function types load one page at a time. If the relationships between detail and header records do not require the EDTTRN or DSPTRN function types, use an EDTFIL or a DSPFIL function type instead.

The PMTRCD function type has less in-built functionality than the Edit Record (EDTRCD) or Display Record (DSPRCD). If you do not require this functionality, PMTRCD is a better choice.

Specifying the Right Level of Relations Checking

CA 2E ensures that all device design relations are satisfied. As a rule of thumb, use no more referential integrity checking than necessary, this includes dropping a relation if it is not being used. You can drop relations either at the access path, field, or function level.

For more information on the types of format relations, see Editing Device Designs in the chapter "Modifying Device Designs."

Action Diagram Editing

For a specific function, you can further adjust relation checking:

- If you specify Optional as the level of relation checking, the relation is enforced if end users enter a value in the field. This means that you do not need to add your own validation to the action diagram. Doing so creates unnecessary processing.
- If you specify No Error as the level of relation checking, CA 2E always checks the relation but issues no error if the relation fails the check.

  Note: The No Error option is useful for distributed applications.
- If you specify User as the level of relation checking, you must add your own validation for the relation in the action diagram.

Construct Resolution in Code

CA 2E-generated code is typically more consistent than custom-created code. However, you can achieve similar functional results with differing action diagram constructs. The constructs result in different source code and object programs, which may have different performance characteristics. This is potentially true of the way CA 2E generates internal functions.

CA 2E implements each reference to an internal function as a different set of code, often inline code. This approach can improve performance. Parameters passed to internal functions are embedded directly in the code at the point of reference, making each instance of the internal function unique.

Using Single Compound Conditions

It is common to repeat a function in a multiple condition CASE structure. For example:

```
.—CASE
|—CTL.Order Header Status is *Open
| internal-function
|—DTL.Order Detail Status is *Unprocessed
| internal-function
|—CTL.Order Value is *LT CTL.Credit Limit
| internal-function
```

Because each internal function is implemented as separate inline code, the code will be repeated, creating a large source module.

However, instead of repeating the function reference, you can use a single compound condition. This eliminates the need to repeat the function references and reduces the number of source lines generated. For example:

```
.—CASE
|— (c1 OR c2 OR c3)
| |— c1 CTL.Order Header Status is *Open
| |— c2 RCD.Order Detail Status is *Unprocessed
| |— c3 CTL.Order Value *LT CTL.Credit Limit
| internal-function
'— ENDCASE
```

Selecting the Proper User Points

When you need to add functionality to an action diagram, study the appropriate function structure chart and information on user points and select the user point to execute at the correct time for your needs.

Using the incorrect user point in an action diagram can make the repetition of code at another user point unnecessary.

Assume the following:

- An EDTFIL is required to maintain Customers
- Customers who have a negative balance need to be highlighted when the EDTFIL presents Customer records to the user
- The WRK context field HIghlight Customer is used to indicate that the Customer record should be highlighted

In order for the appropriate Customer records to be highlighted when the records are initially loaded, the following processing could be placed in the USER: Initialize subfile record (existing record) user point:

```
.  > USER: Initialize subfile record ( existing record)
. .—
. . WRK.Highlight Customer = CND.No
. . .-CASE
. . | -RCD.Customer balance is LT 0
. . |  WRK.Highlight Customer = CND.Yes
. . '-ENDCASE
. '—
```

To further ensure that the records continue to be highlighted after the records are loaded, the same processing would also need to be inserted in the USER: Validate subfile record fields user point.

The duplication of logic in both these user points can be avoided by placing the processing in the USER: Validate subfile record relations user point. This user point is executed both at function load and later when the records are revalidated.

The selection of this user point reduces the amount of code generated thereby improving the efficiency of the resulting program.

Chapter 16: Creating Wrappers to Reuse Business Logic

CA 2E lets you easily retrieve user-written business logic, such as validation routines, and place them into separate functions by using wrappers. These functions can then be accessed by other CA 2E functions or by external procedures such as CA Plex functions.

The process has two parts:

1. Select the action diagram statements that you want to place into other functions.
2. Select a function type and name. An automated process copies the statements and places them in a new function with an automatically generated parameter interface.

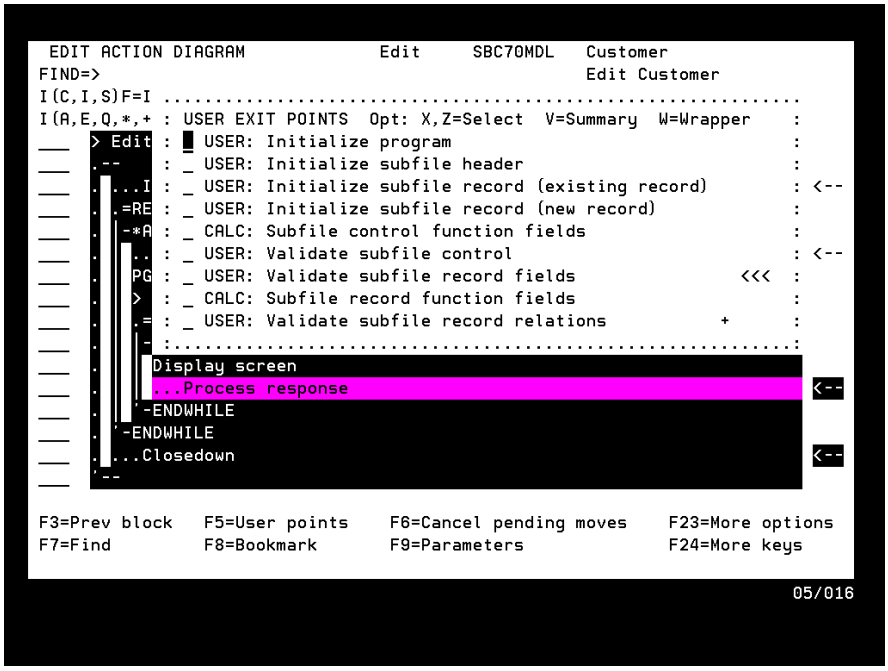
This section contains the following topics:

Selecting Action Diagram Statements

You can select the action diagram statements from a User Point or from the Notepad.

From a User Point

1. While editing the action diagram of a function, press F5 to view the User Exit Points.
2. Enter **W** next to the User Point that contains all the statements you want to place in another function by using a wrapper.



3. Go to the section Selecting Function Name and Type.

From the Notepad

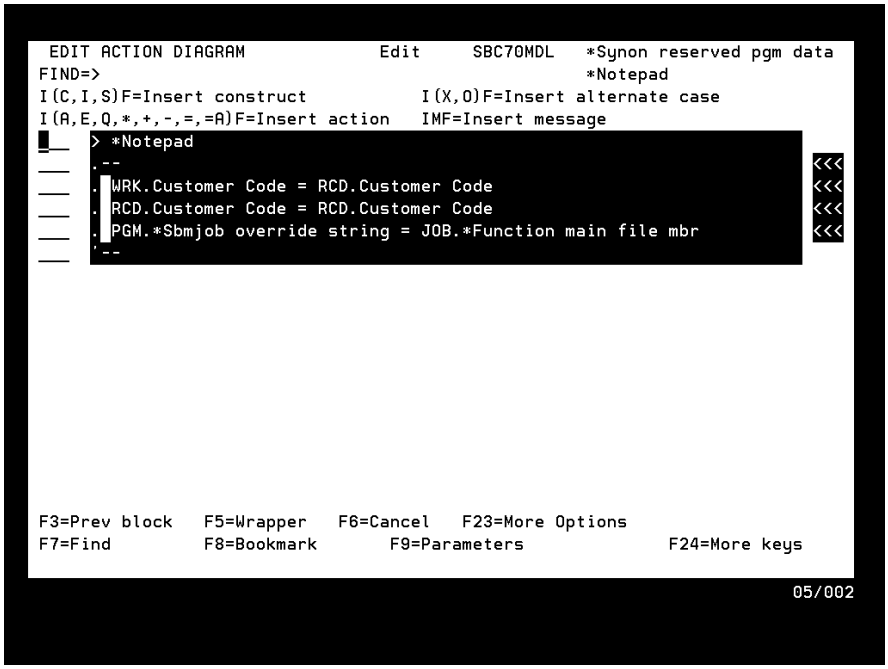
In some cases, it may not be desirable or necessary to use a wrapper on an entire User Point. The User Point may contain statements that you do not want in the new function. Or perhaps the final function needs additional statements that are not required in the original function.

You can edit User Point action diagram statements, place the whole User Point in a wrapper, and then exit the function without saving to preserve the original User Point. An easier and more flexible approach, however, is using the Action Diagram Editor Notepad.

Note: Because EXCEXTFUN and EXCINTFUN have no User Points, use Notepad to place statements from these types of functions in a wrapper.

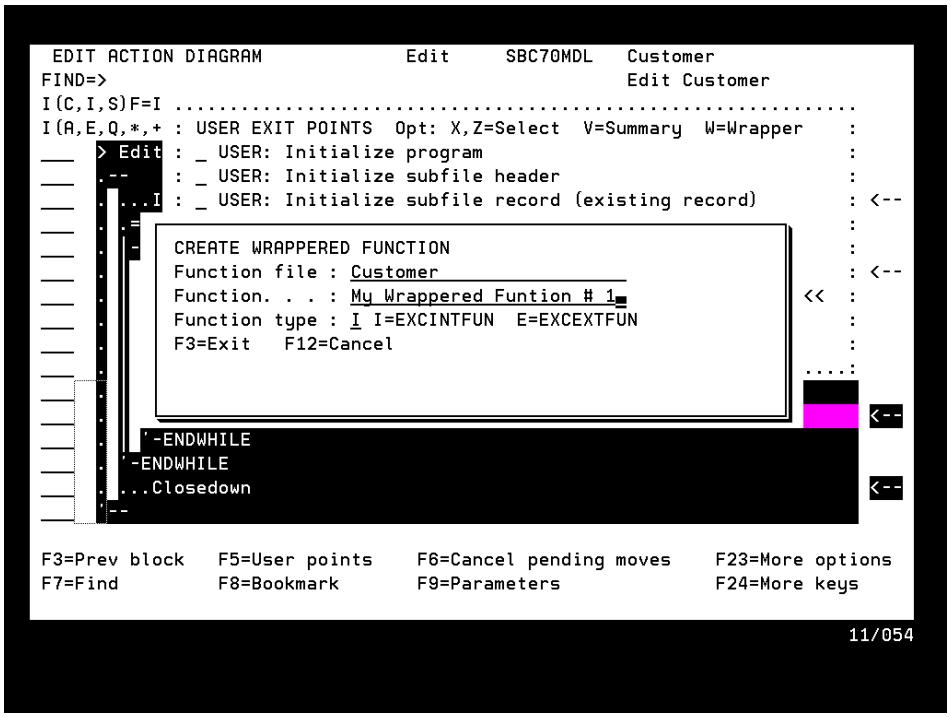
Use the Notepad commands N, NA, NAA, NI, NR, and NRR to put the required statements into the Notepad.

1. Press F18 to view the *Notepad.
2. Press F5 to select all the statements in the Notepad.



Selecting Function Name and Type

After selecting the action diagram statements for the wrapper, you must assign a function name and type.



The default values that appear in the Create Wrappered Function panel depend on whether you started the procedure from a User Point or from the Notepad:

From a User Point

The default Function File, which will own the new function, is the same file that owns the function that contained the User Point. The default Function name is the first 25 characters of the User Point name.

From the Notepad

The default Function File is ?, therefore you must choose the function name. The default Function name is blank.

In either case, the default Function type is E for EXCEXTFUN, but you can change it to I to create an EXCINTFUN.

The final step is to press Enter. An Execute External Function or Execute Internal Function is created. The selected statements are copied from the Notepad or the User Point into the newly created function.

Automatic Parameter Interface Generation

The function placed in a wrapper creates parameters automatically based on the field contexts used in the original action diagram statements. Some contexts do not require conversion; these contexts are LCL, NLL, ARR, PGM, JOB, and CON.

All other contexts are converted and passed into the new function as duplicate parameters. EXCEXTFUN and EXCINTFUN functions do not have associated screens or database fields, so the action diagram cannot refer to contexts such as RCD, CTL, DB1, and DB2.

For each field in an unavailable context, the wrapping process creates an entry on an array. A new array is created for each function placed in a wrapper. This array is then defined on the parameter listing for the function passed as RCD.

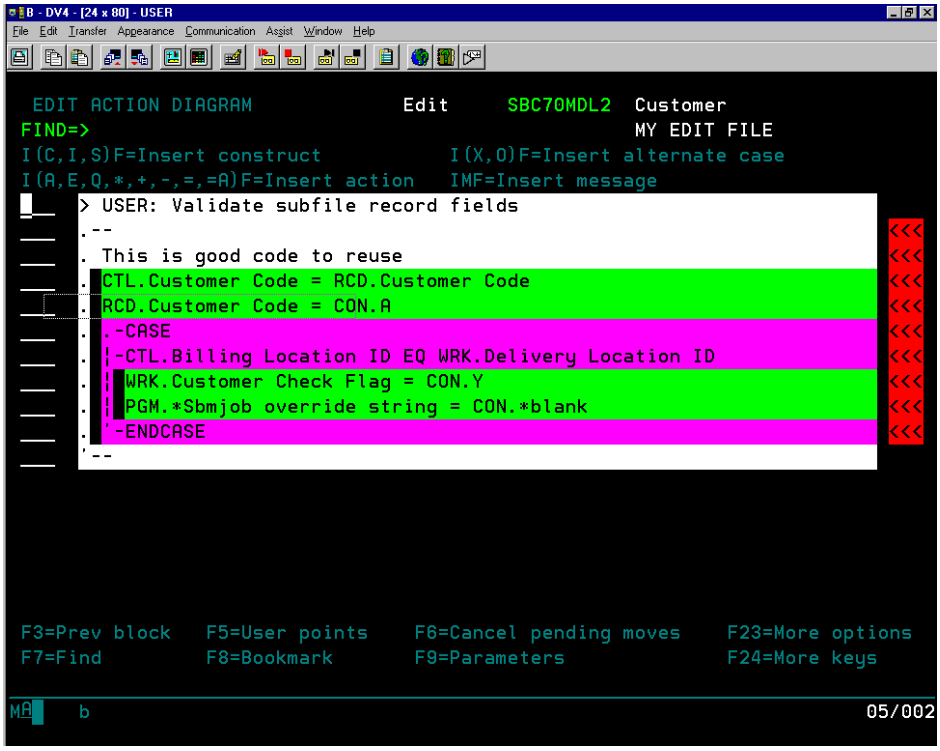
Each new unavailable context is associated with another parameter entry of the same array. Each array is passed as a duplicate parameter context, from PR1 to PR9. The first unavailable context on the action diagram statements is assigned to PR1. The next context not already assigned to the array is passed as the PR2 parameter context, and so on. LCL, NLL, PGM, JOB, and CON are never substituted.

The parameter usage for each field on each parameter listing is calculated from how the CTX field is used in the action diagram statements of the function.

The following illustrations show how the field contexts are converted to the PR1 to PR9 contexts.

Original Contexts

Edit Customer is a function of the type EDTFIL.



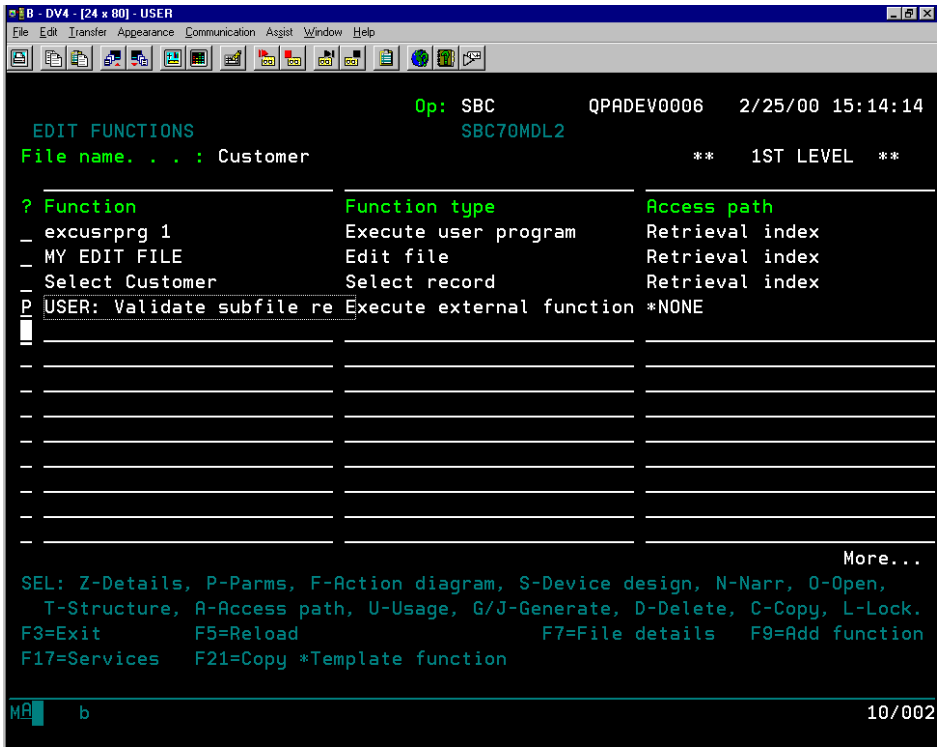
In this example, the code in USER: Validate subfile record fields is placed in a wrapper. This example shows parameter substitution, not application design.

This code populates CTL.Customer Code with RCD.Customer Code, sets RCD.Customer Code to CON.A, and checks whether CTL.Billing Location is equal to WRK.Delivery Location. If it is, WRK.Customer Check Flag is set to Y and PGM.*Sbmjob receives the override string CON.*BLANK.

1. Press F5 to select User Points, and enter **W** for the relevant one.
2. Choose the function name Update Override String.

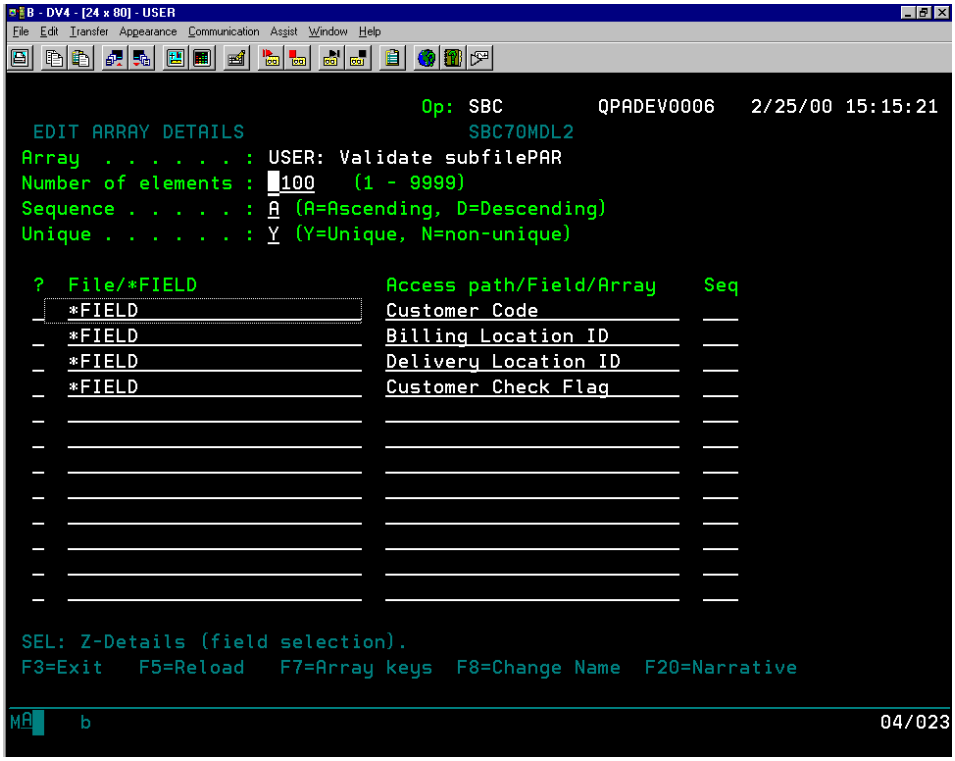
The function and the parameter interface are created and contexts are substituted for any contexts unavailable to the EXCEXTFUN. CTL is the first unavailable context, so an array is created for this function. The array name consists of the first 22 characters of the function name plus PAR at the end.

The Newly Created Function



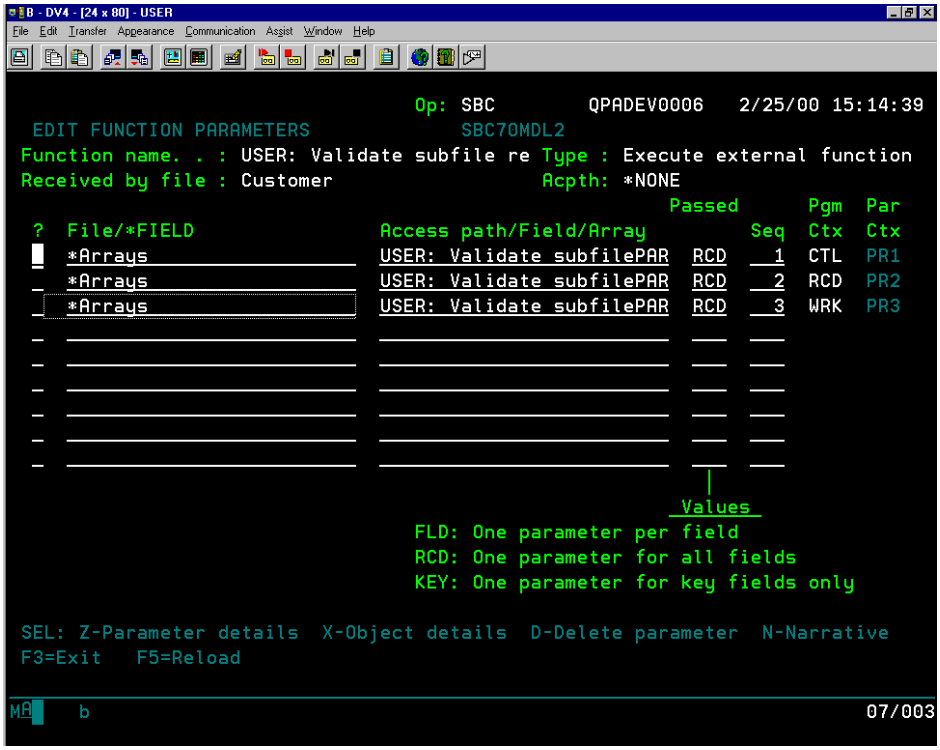
The Newly Created Array

A new array "USER: Validate subfile PAR" is created. It contains an entry for each field in the original function in a context that is not available in the newly created EXCEXTFUN.



The Parameter Definitions

The array then establishes parameters on the created function. It is passed once for each context that is being substituted. The Pgm Ctx and Par Ctx columns show the original and substituted contexts.

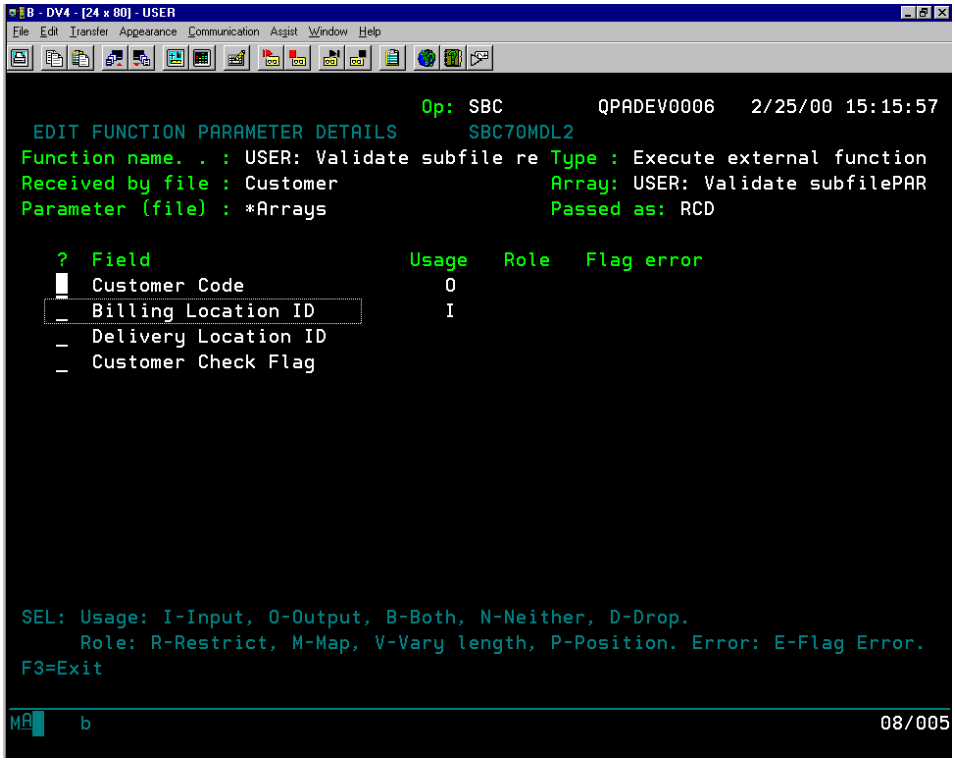


The Control Context

Two fields were used in the CTL context in the original action diagram: Customer Code and Billing Location ID. Thus both fields are specified as parameters for the first array entry.

Notice how the usage matches the usage of the fields in the original action diagram. If the field was used as a target of an operation, the usage defaults to Output. If the field was used as input to an operation, the usage defaults to Input.

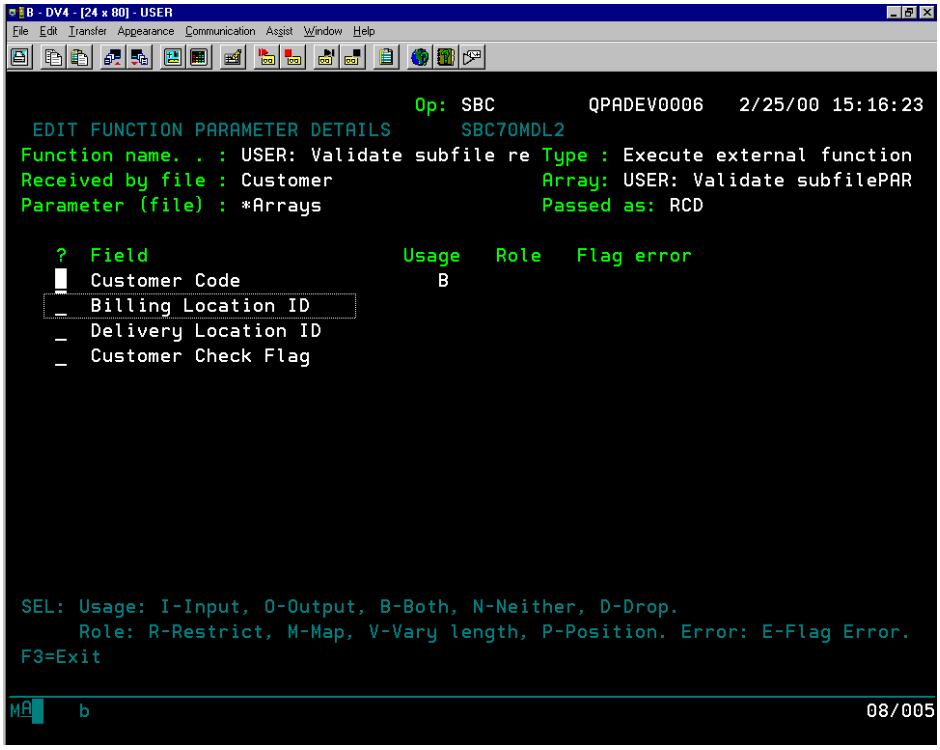
After conversion, the former CTL context is now the PR1 context.



The Record Context

For the RCD context entry, only one field is specified as a parameter. The usage B (Both) matches the usage of the field in the original action diagram. The field Customer Code was used as both input and as the target of a *Move statement.

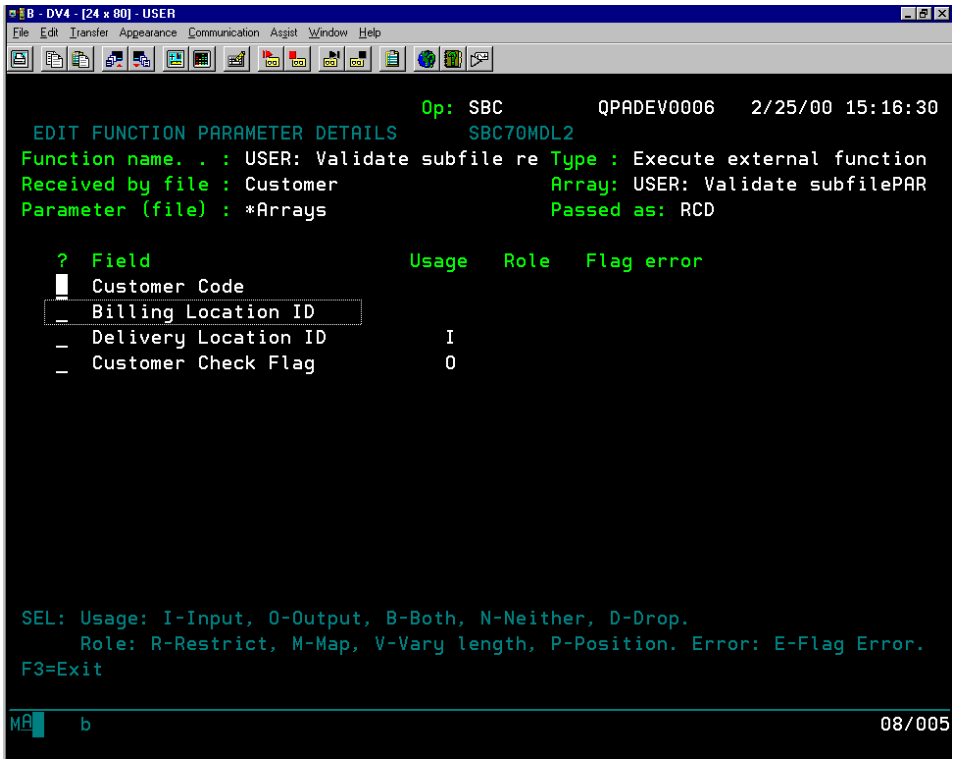
After conversion, the former RCD context is now the PR2 context.



The WRK Context

The WRK context entry matches the usage of the field in the original action diagram statements.

After conversion, the former WRK context is now the PR3 context.

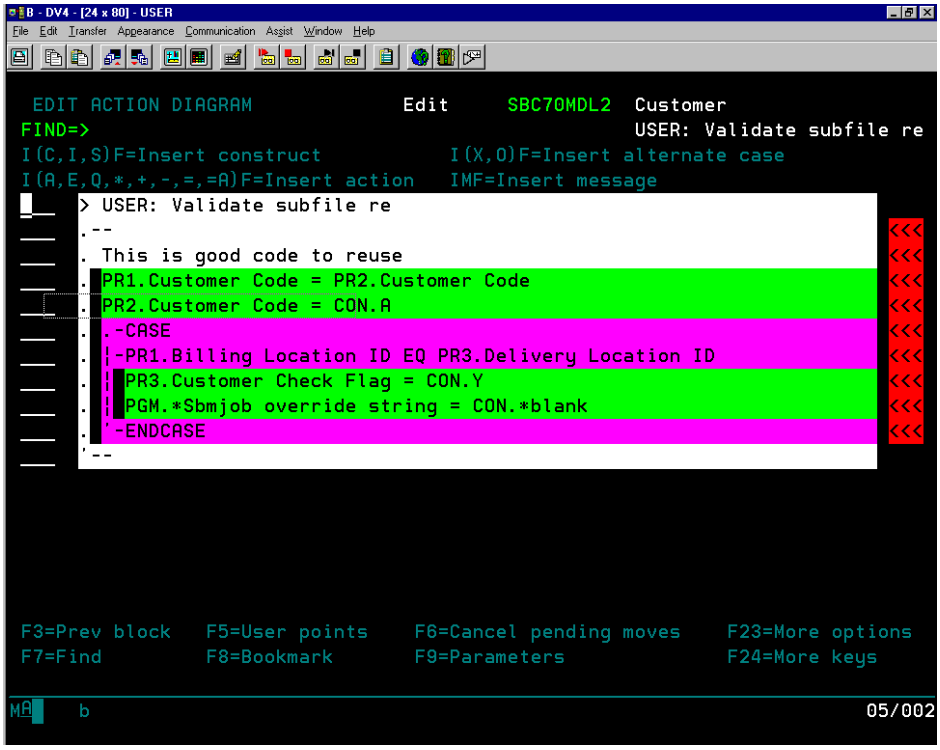


The New Action Diagram

The new action diagram looks like the old one except for the substituted contexts.

If a reference to an RCD context had appeared first, instead of the CTL context, then the order of substitution would be different. PR1 would have been RCD.

The newly created function can now be used and called in the usual manner.



Appendix A: Function Structure Charts

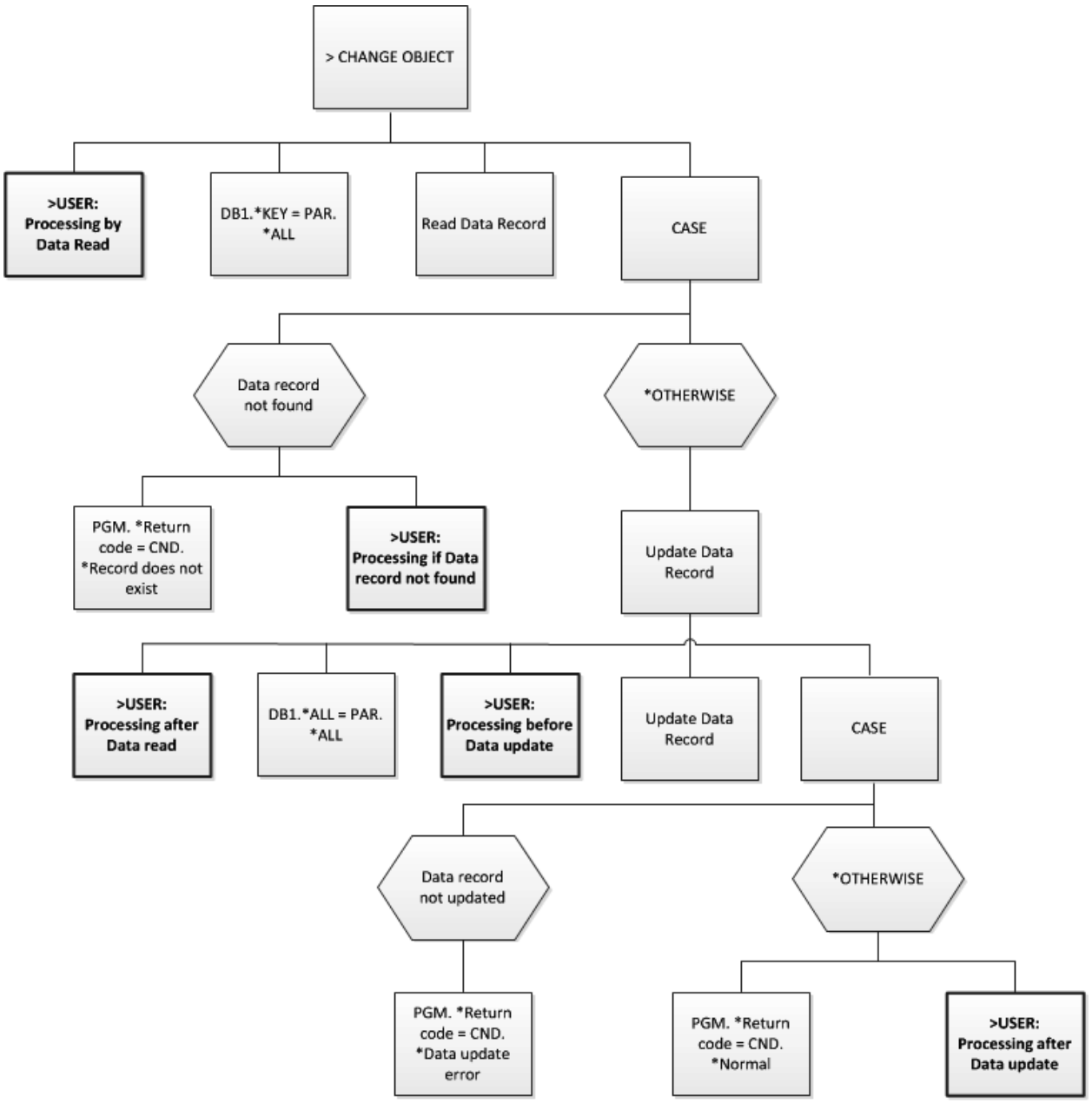
This appendix contains examples of the CA 2E function structure charts. The structure charts provide you with a visual orientation of the user points with respect to other processes. They will help you learn the processing offered by each function type. Work your way through each chart as necessary. Evaluate each user point with respect to the rest of the function until you locate the correct point at which you want to introduce your processing logic.

Each structure chart is designed to be read from top to bottom and from left to right. The user points are shown in boxes outlined in bold. A box with a bold outline in its lower right corner indicates that the chart continues on the next page.

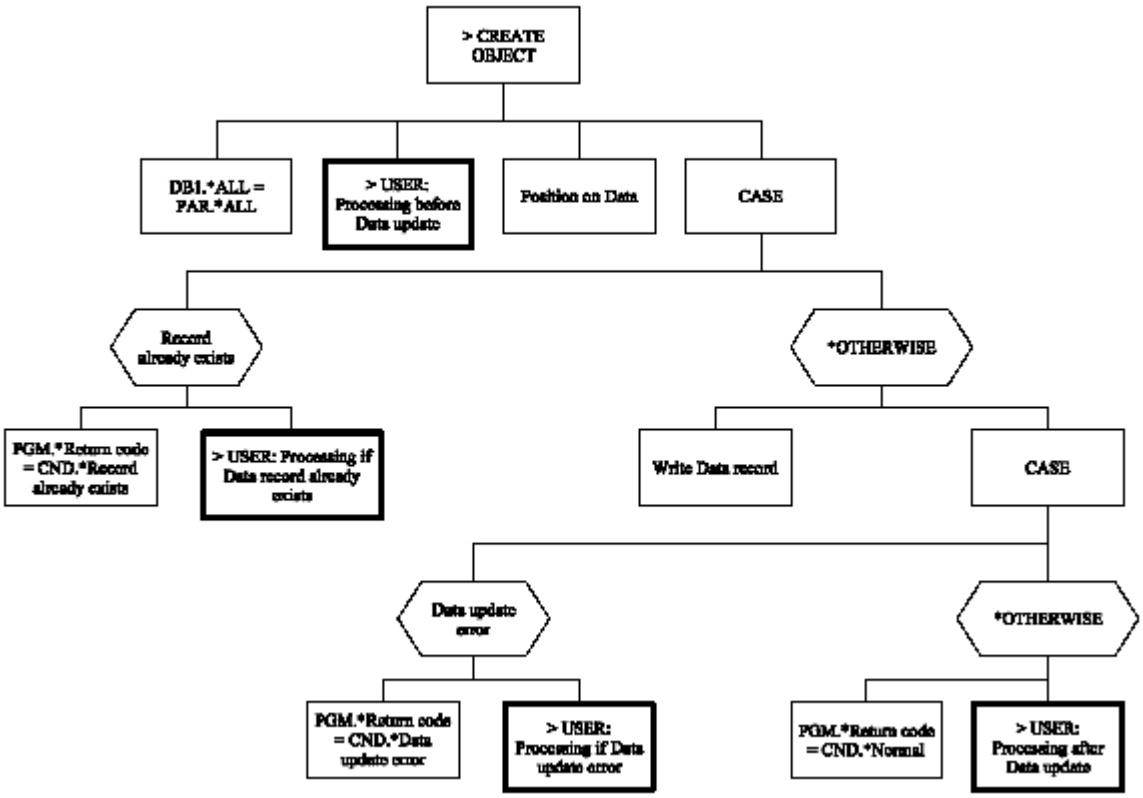
For more information on user points, see Understanding Action Diagram User Points in the chapter “Modifying Action Diagrams.”

The structure charts are shown on the following pages in alphabetical order by function type.

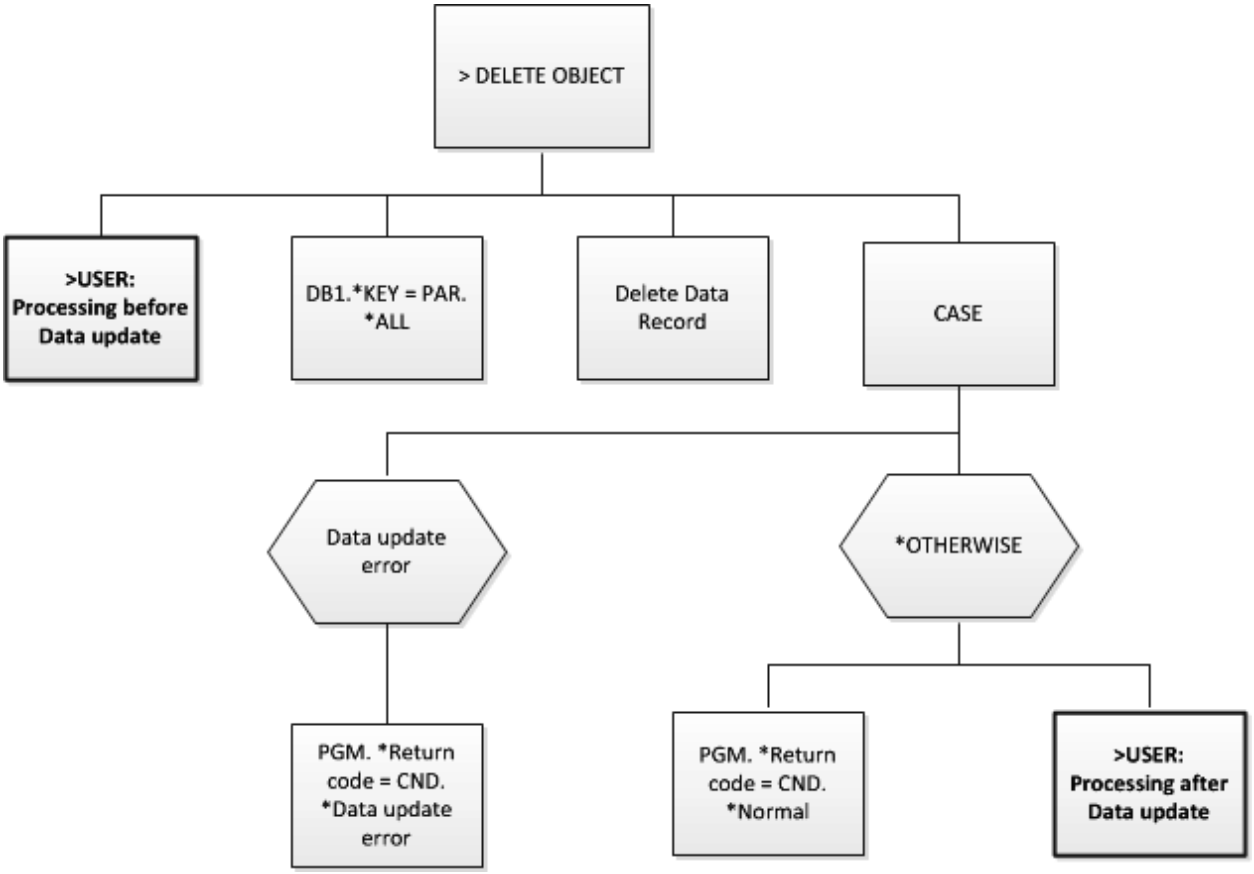
Change Object



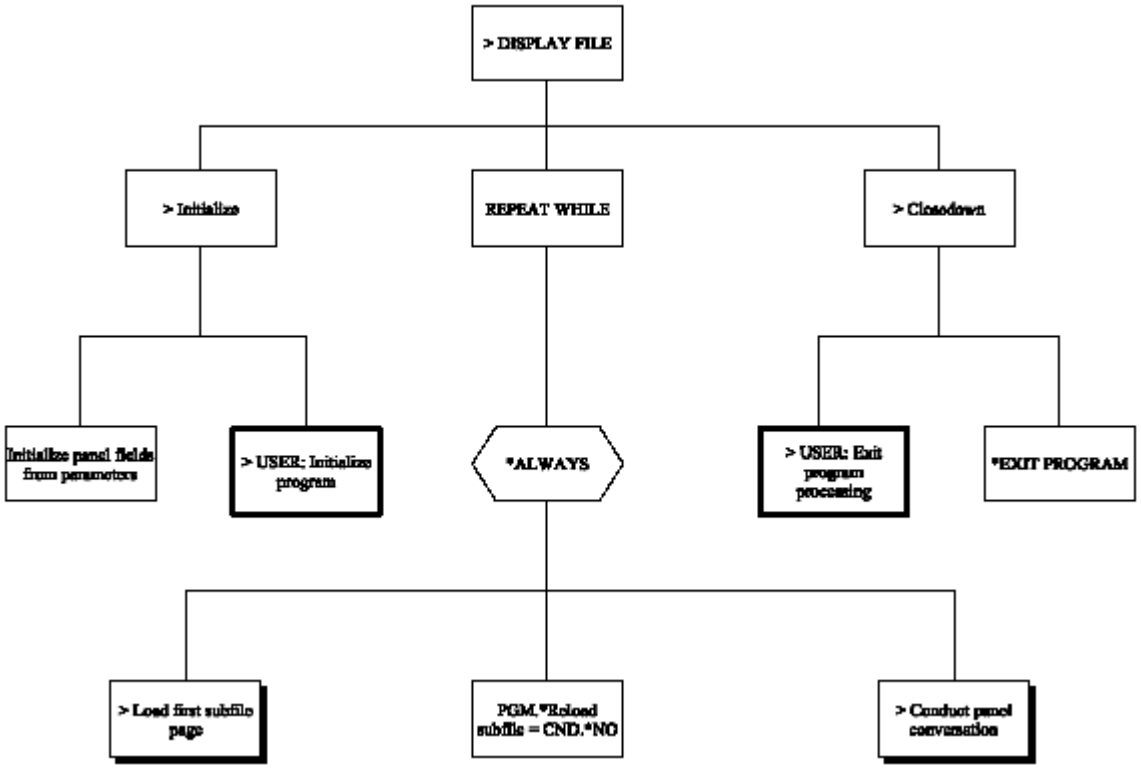
Create Object



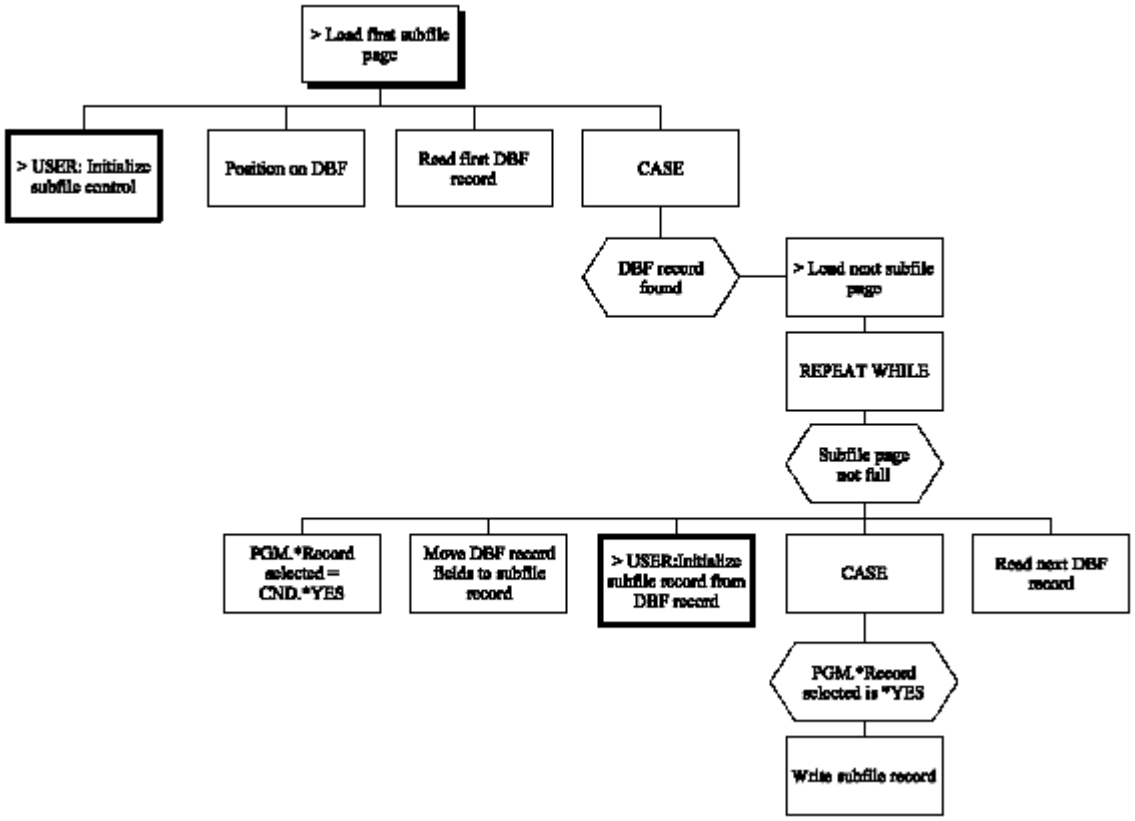
Delete Object



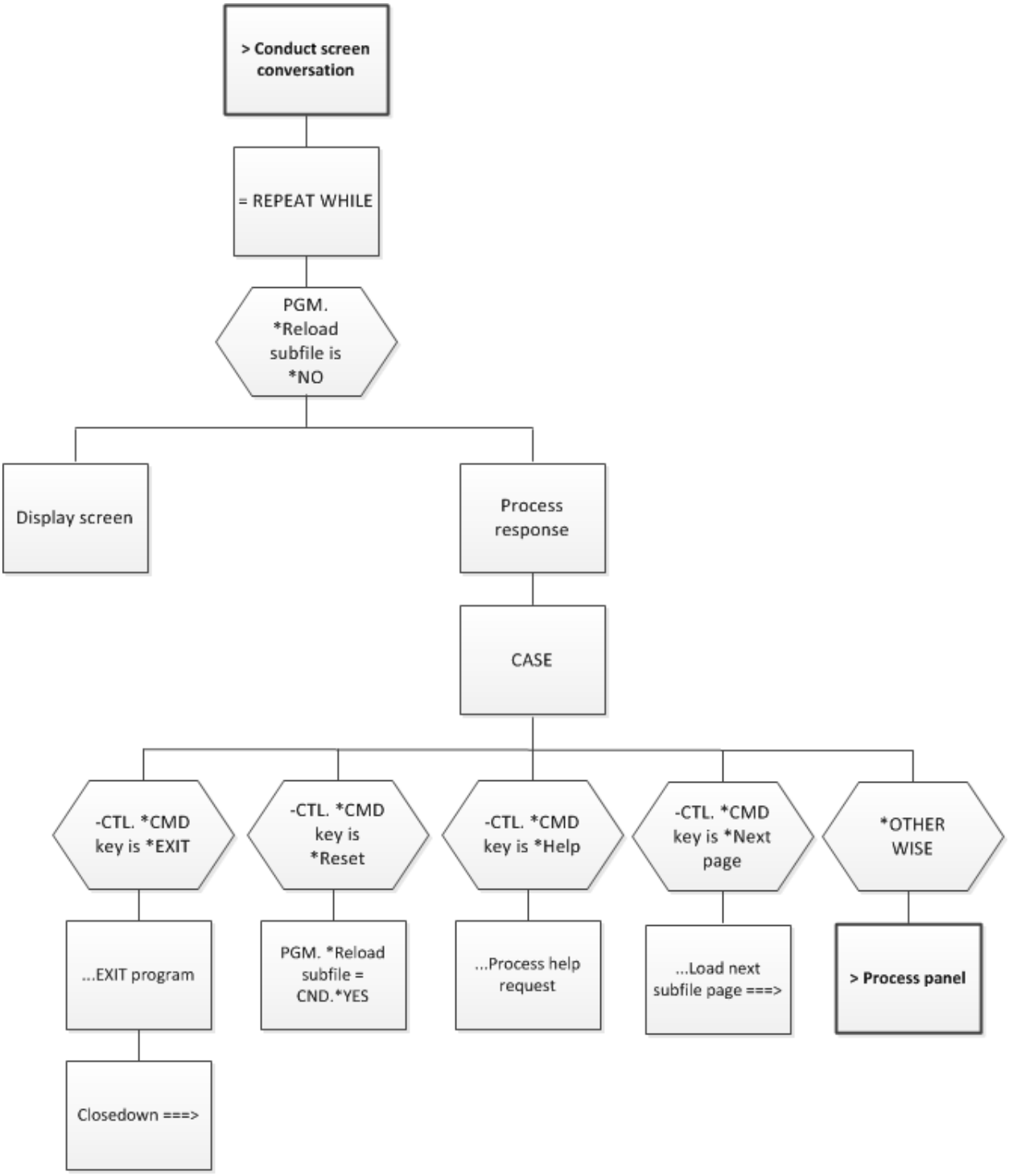
Display File (Chart 1 of 5)



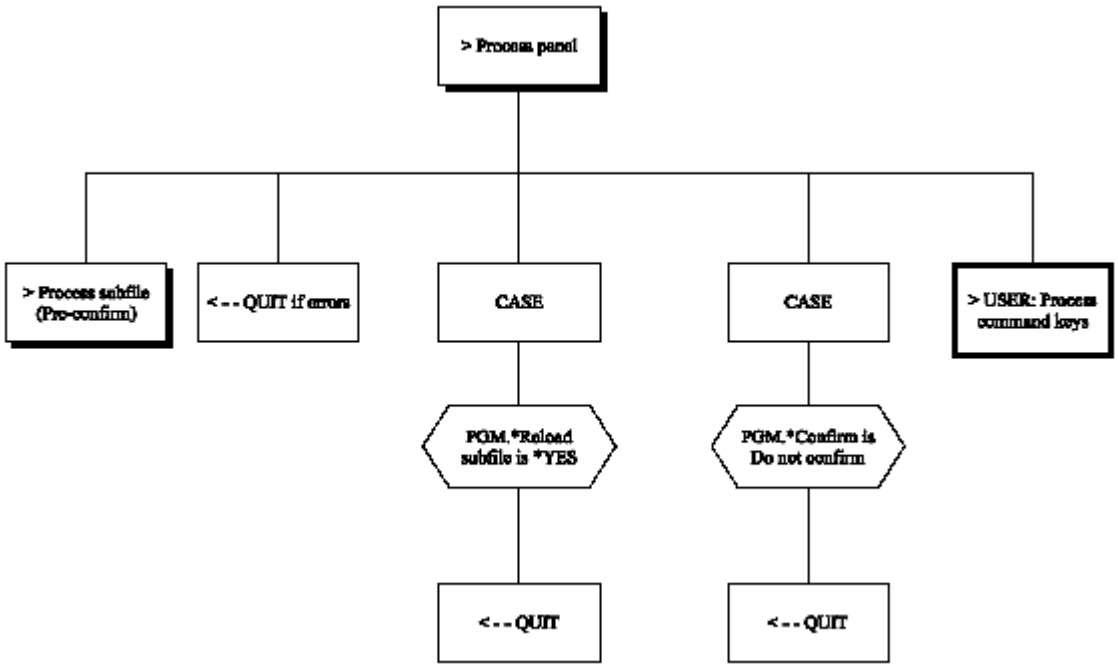
Display File (Chart 2 of 5)



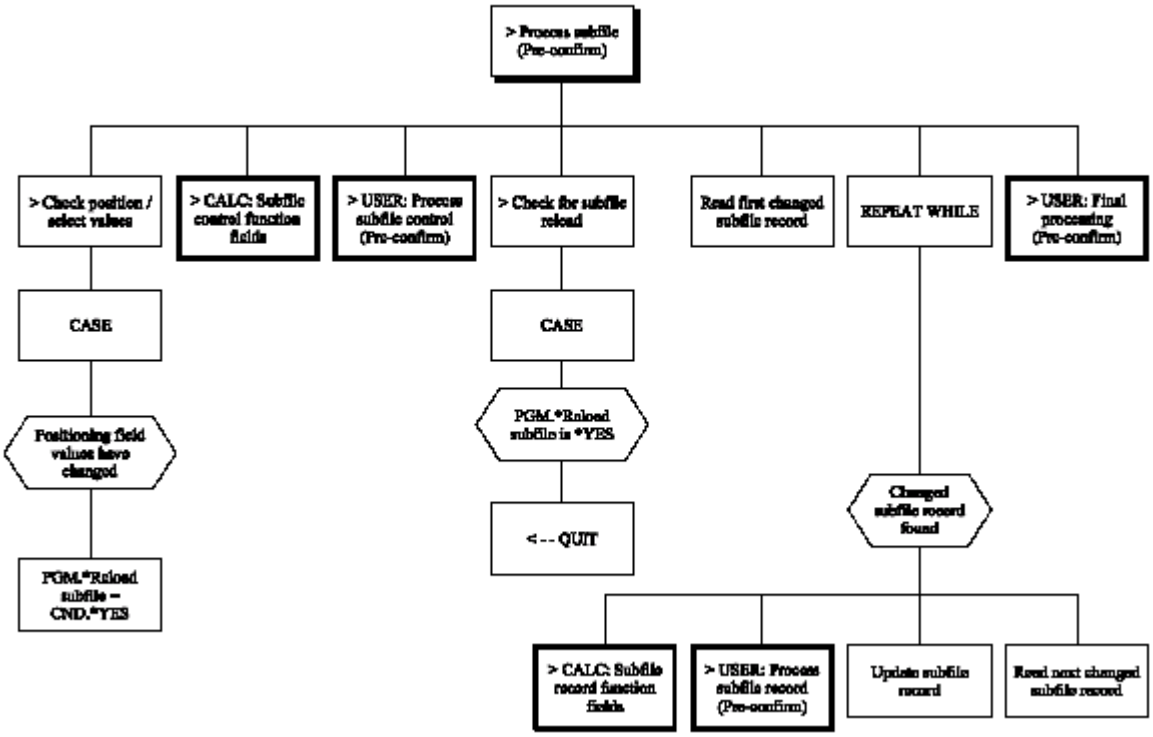
Display File (Chart 3 of 5)



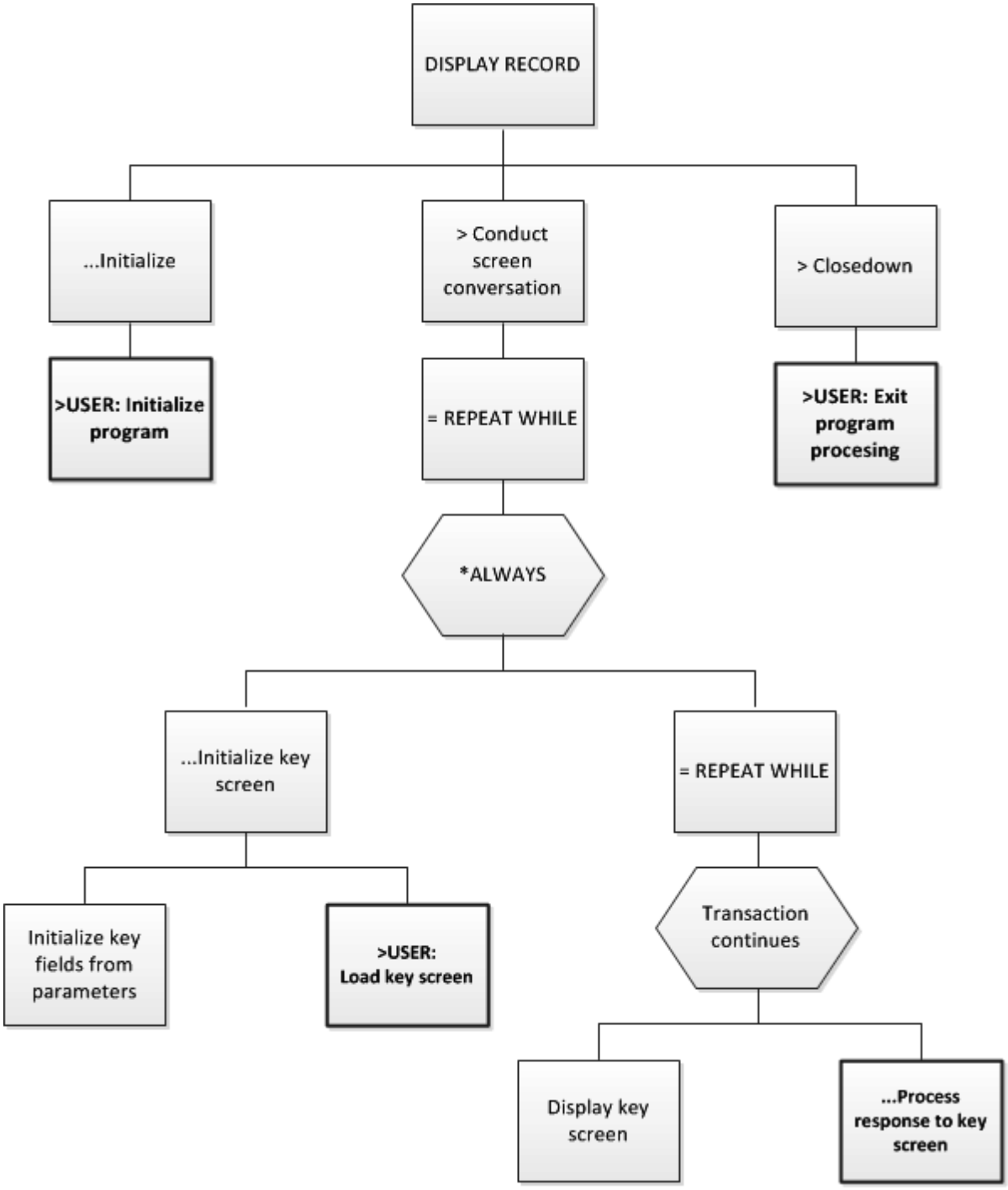
Display File (Chart 4 of 5)



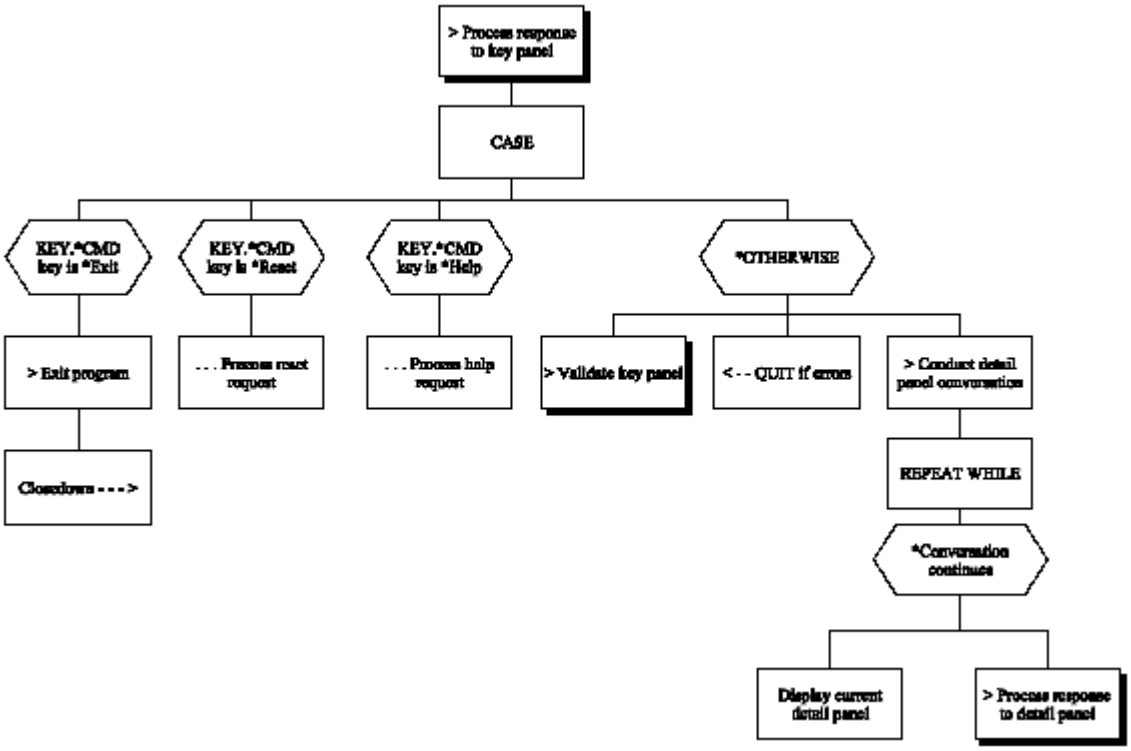
Display File (Chart 5 of 5)



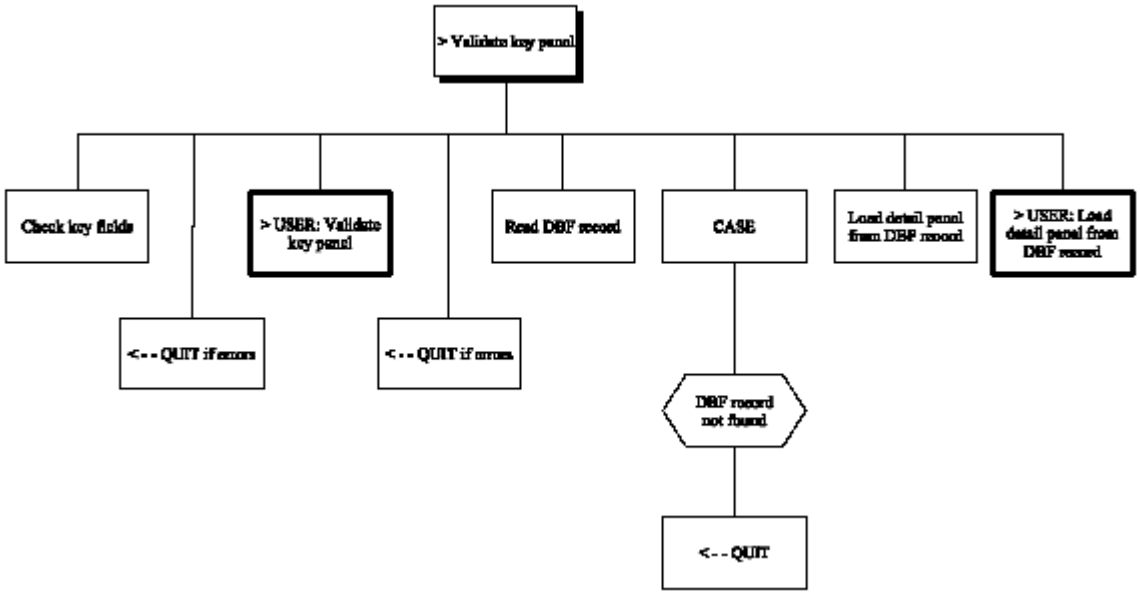
Display Record (Chart 1 of 5)



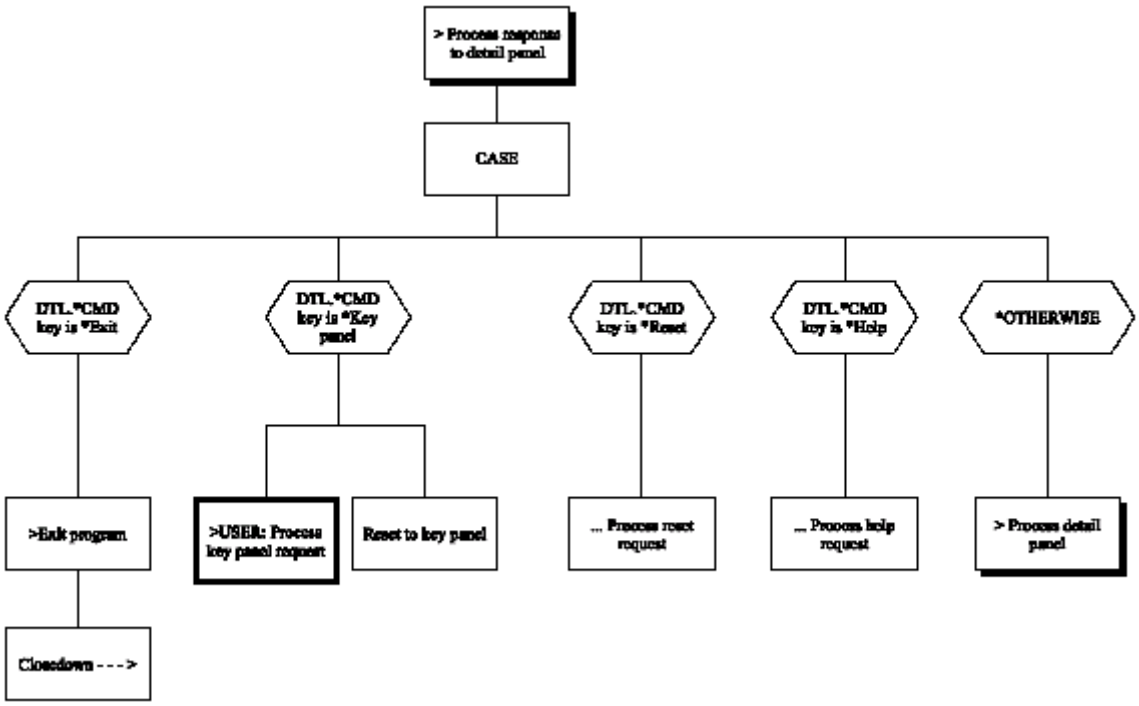
Display Record (Chart 2 of 5)



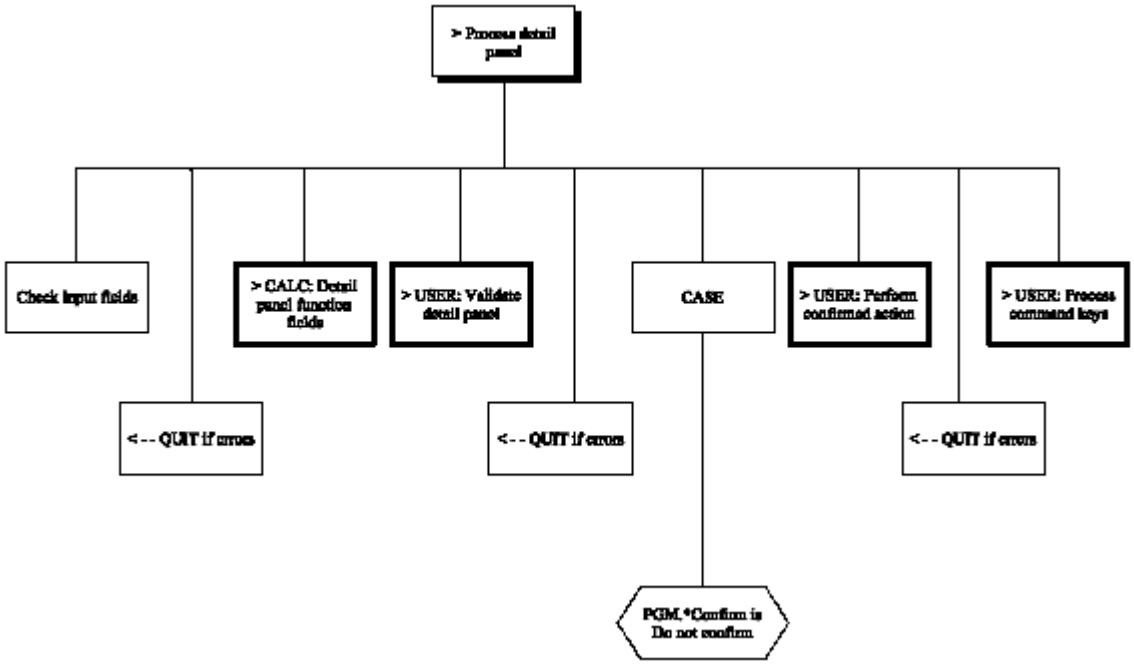
Display Record (Chart 3 of 5)



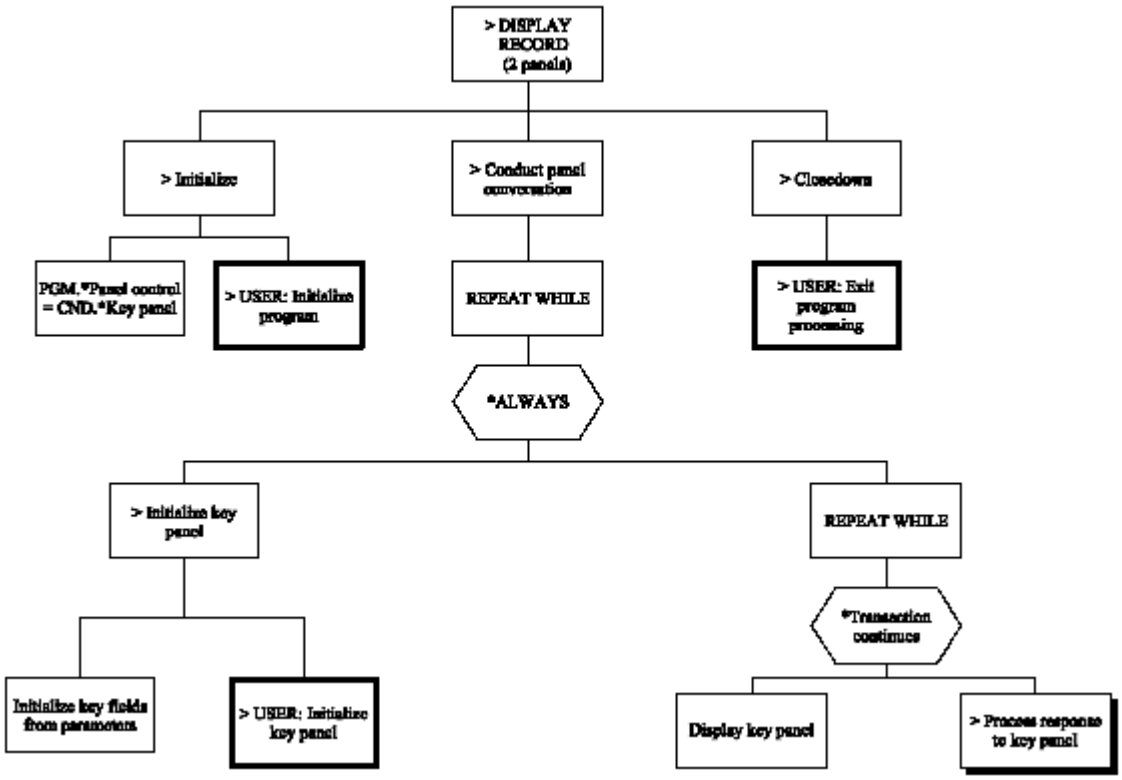
Display Record (Chart 4 of 5)



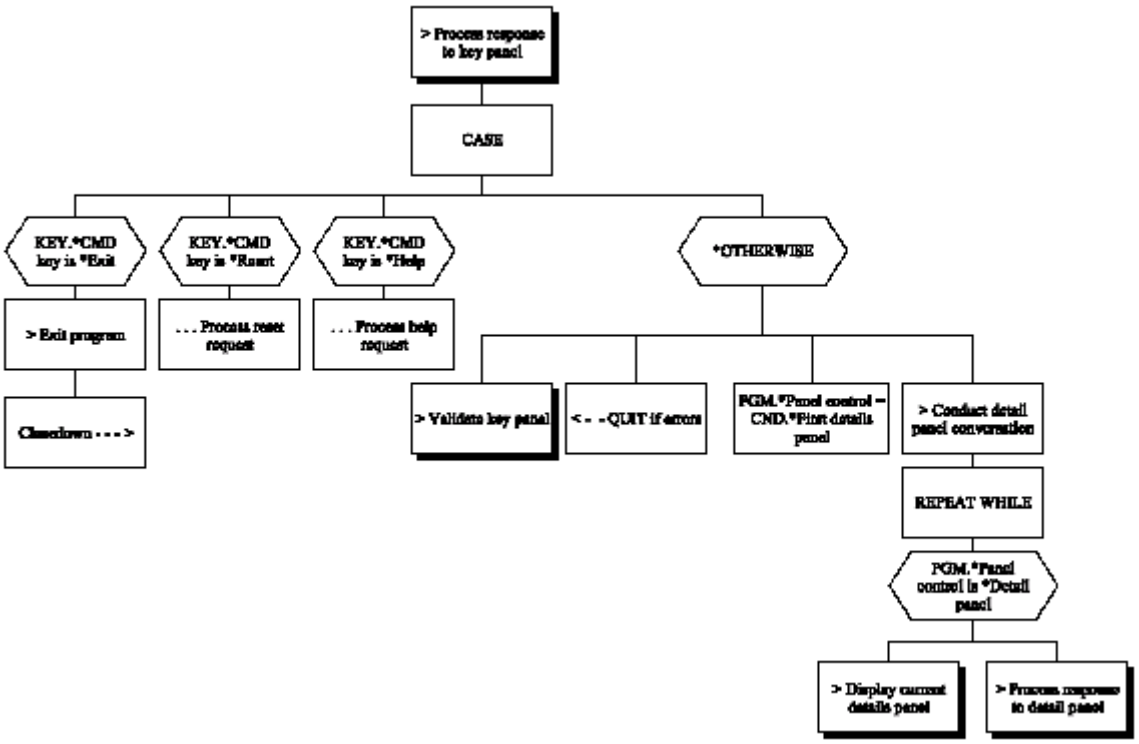
Display Record (Chart 5 of 5)



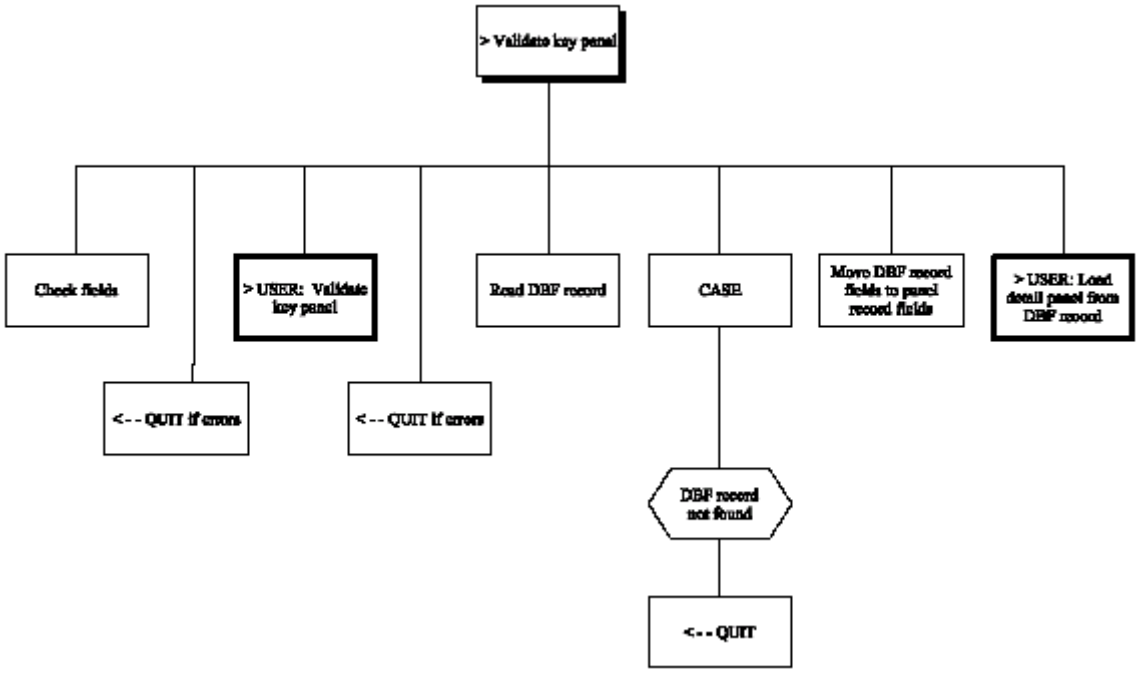
Display Record– 2 Panels (Chart 1 of 7)



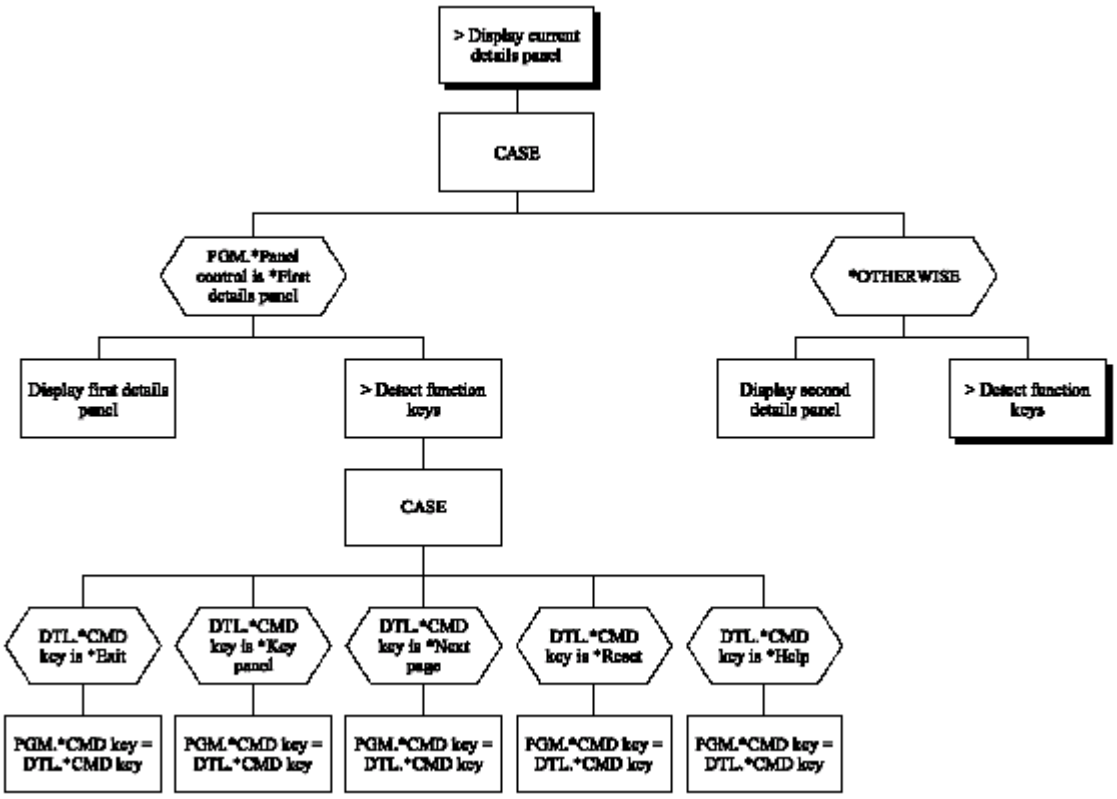
Display Record – 2 Panels (Chart 2 of 7)



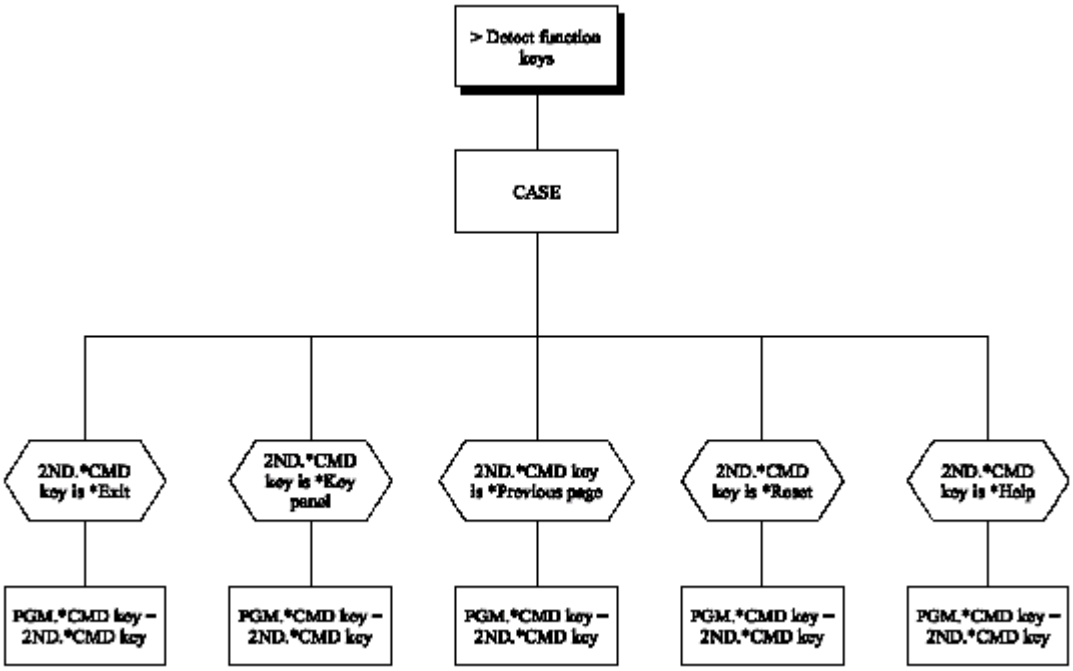
Display Record – 2 Panels (Chart 3 of 7)



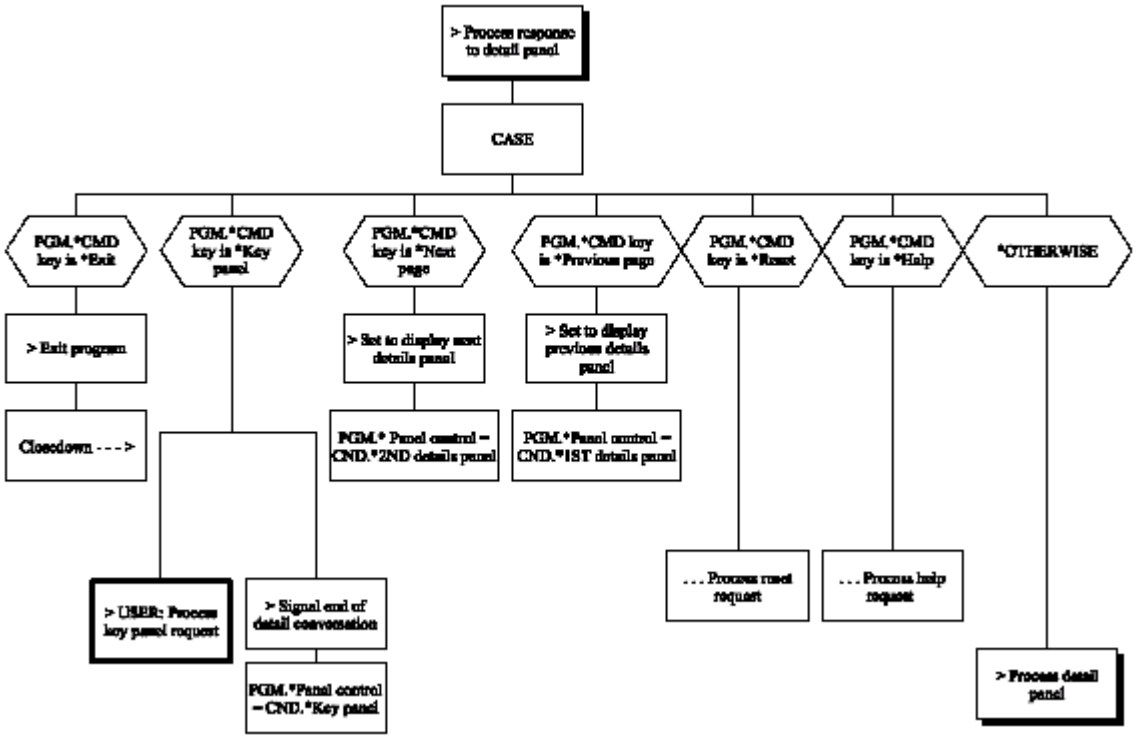
Display Record – 2 Panels (Chart 4 of 7)



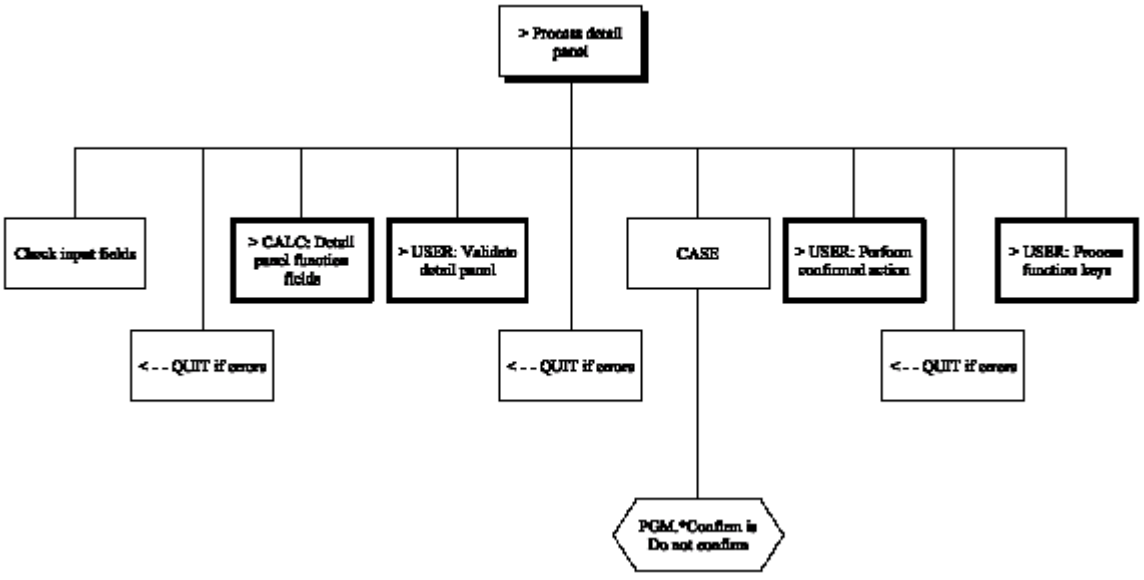
Display Record – 2 Panels (Chart 5 of 7)



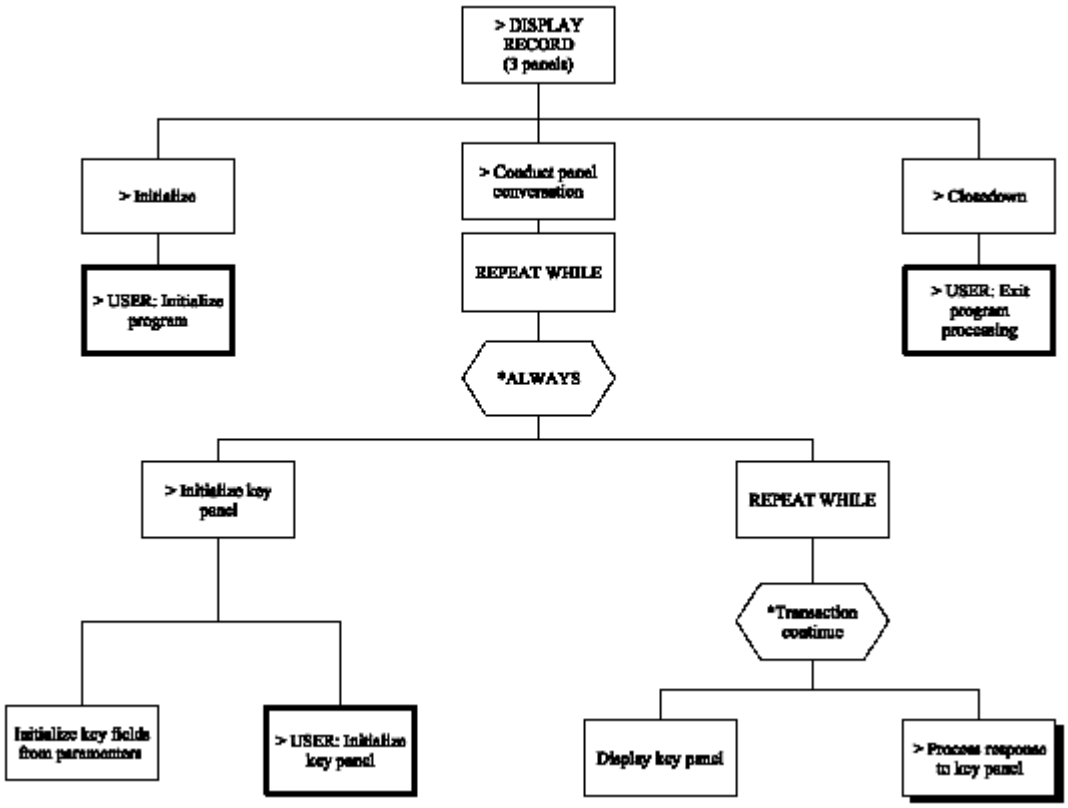
Display Record – 2 Panels (Chart 6 of 7)



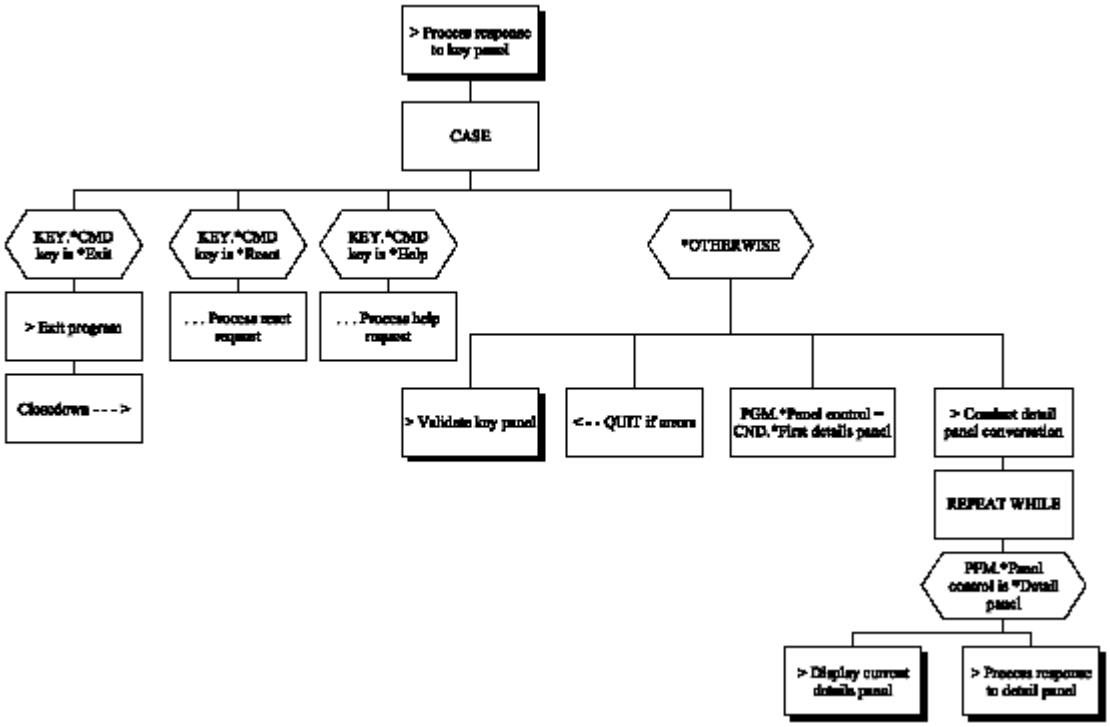
Display Record – 2 Panels (Chart 7 of 7)



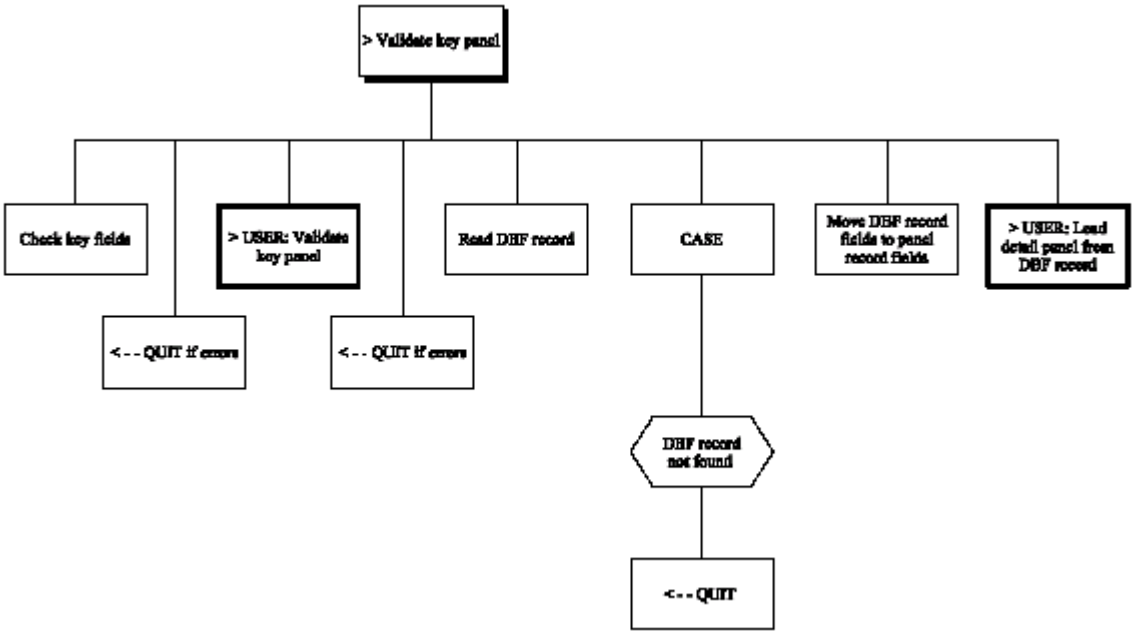
Display Record – 3 Panels (Chart 1 of 8)



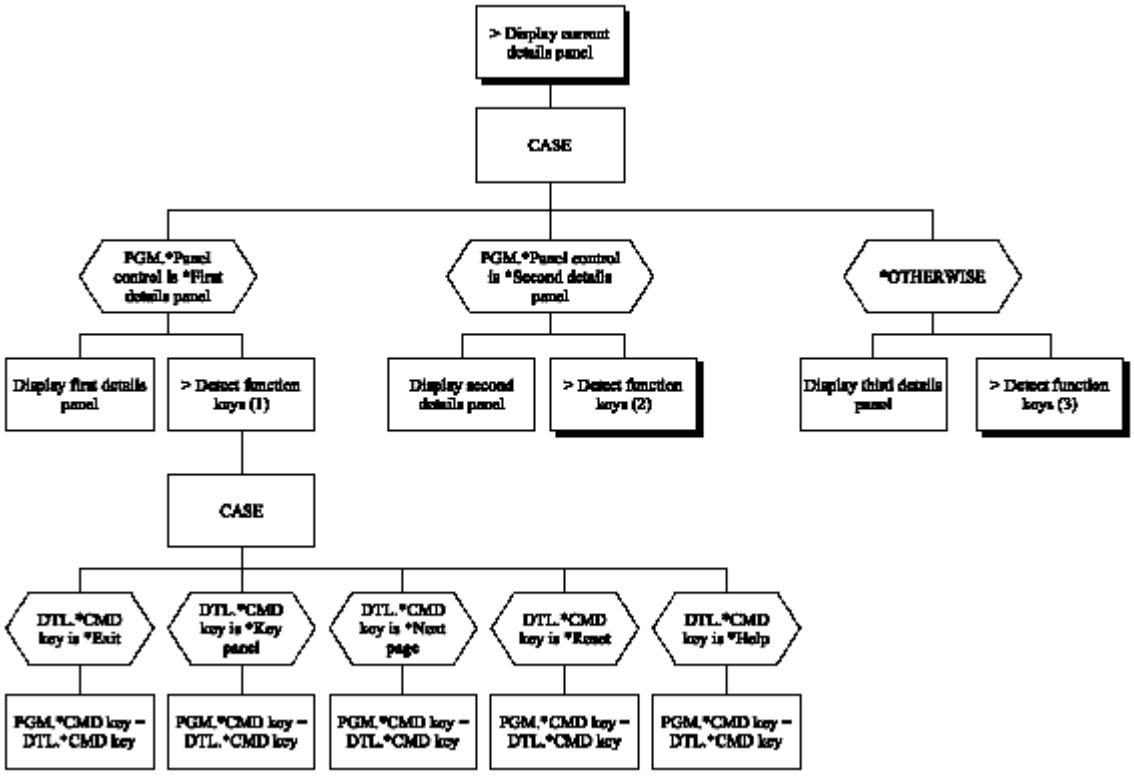
Display Record – 3 Panels (Chart 2 of 8)



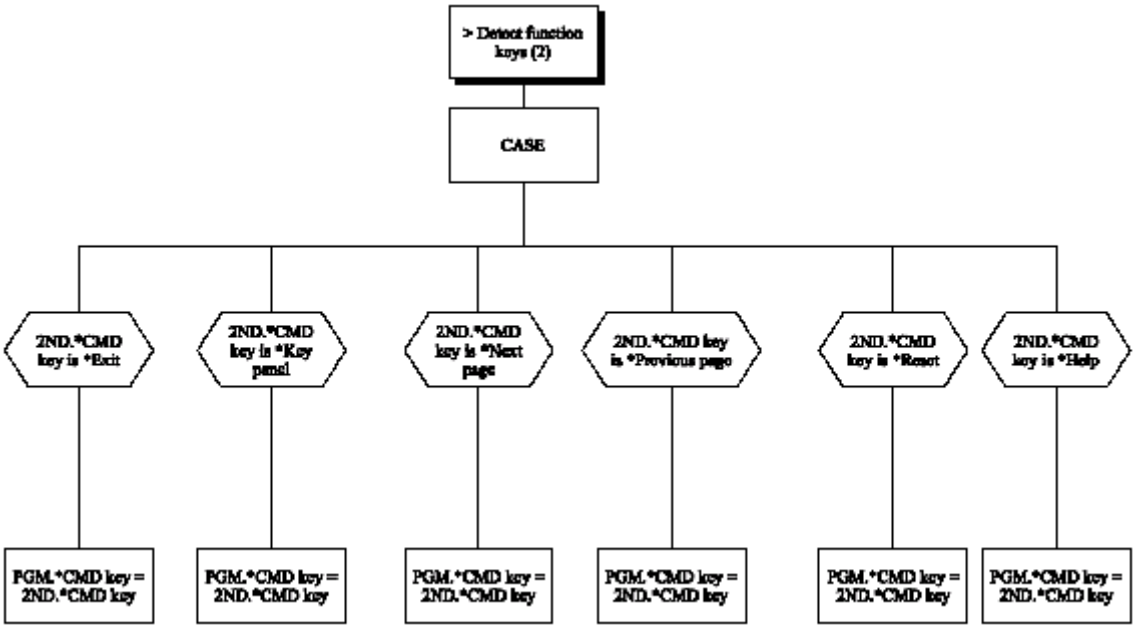
Display Record – 3 Panels (Chart 3 of 8)



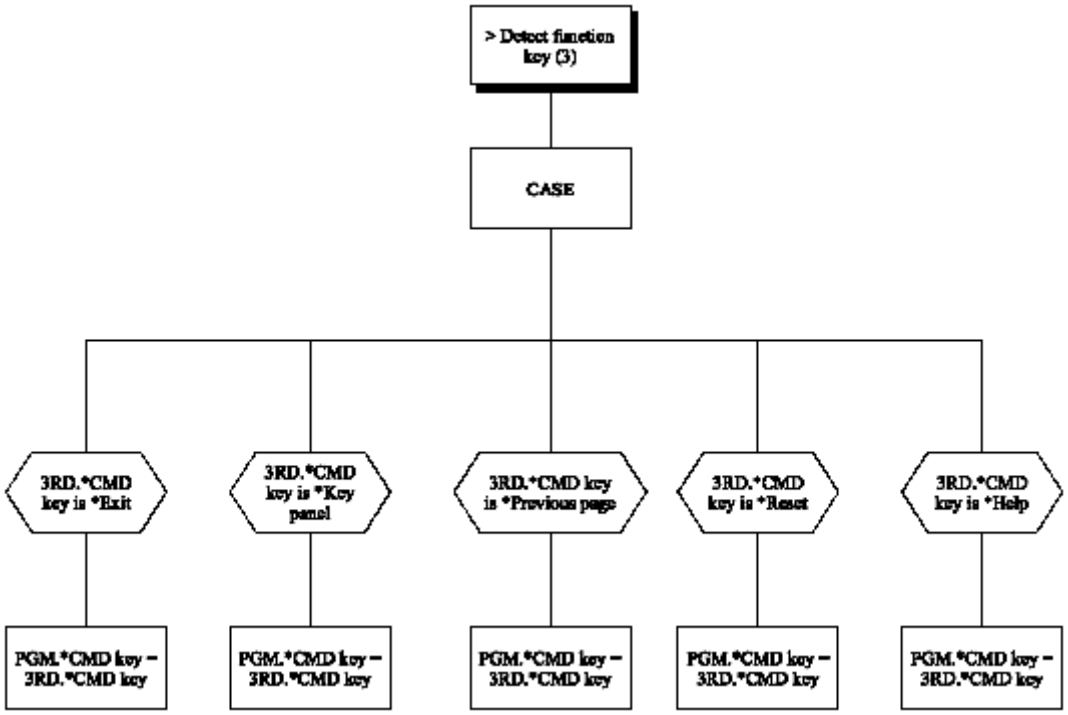
Display Record – 3 Panels (Chart 4 of 8)



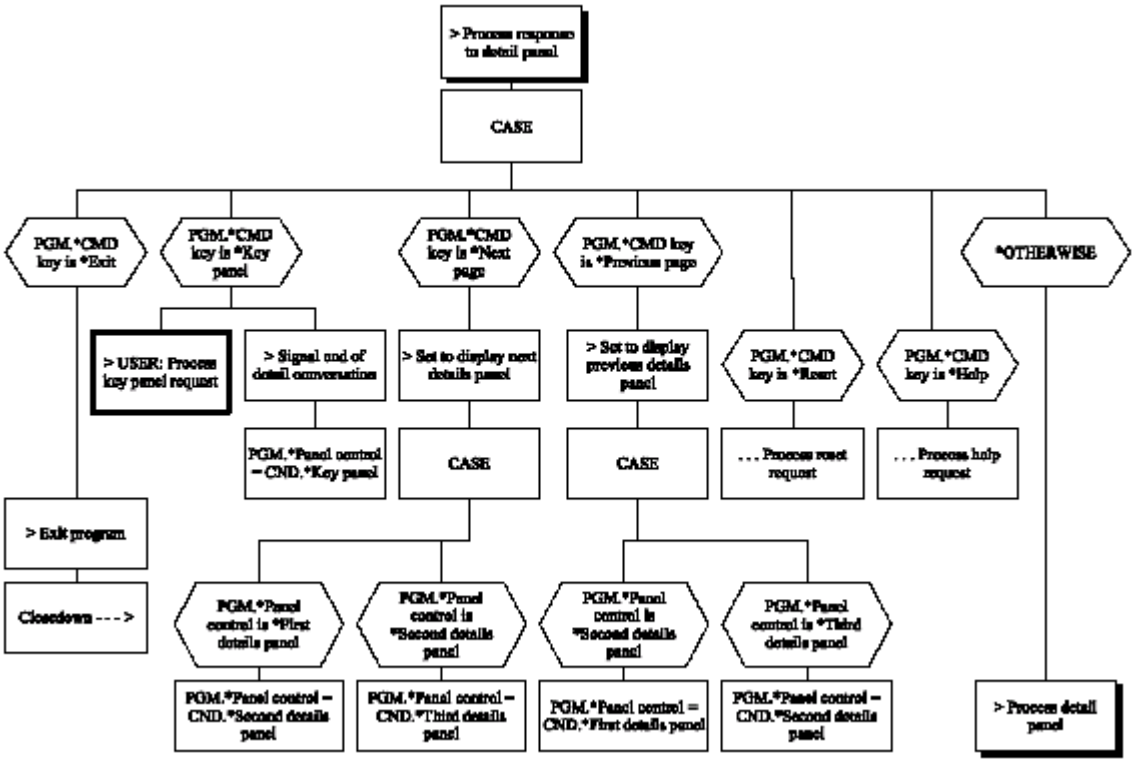
Display Record – 3 Panels (Chart 5 of 8)



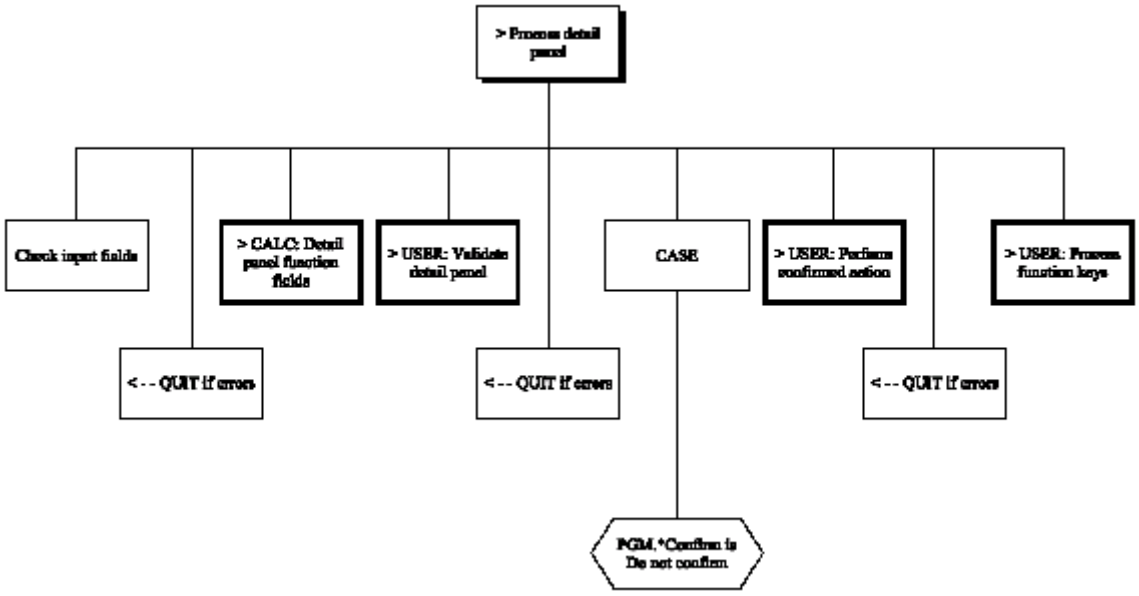
Display Record – 3 Panels (Chart 6 of 8)



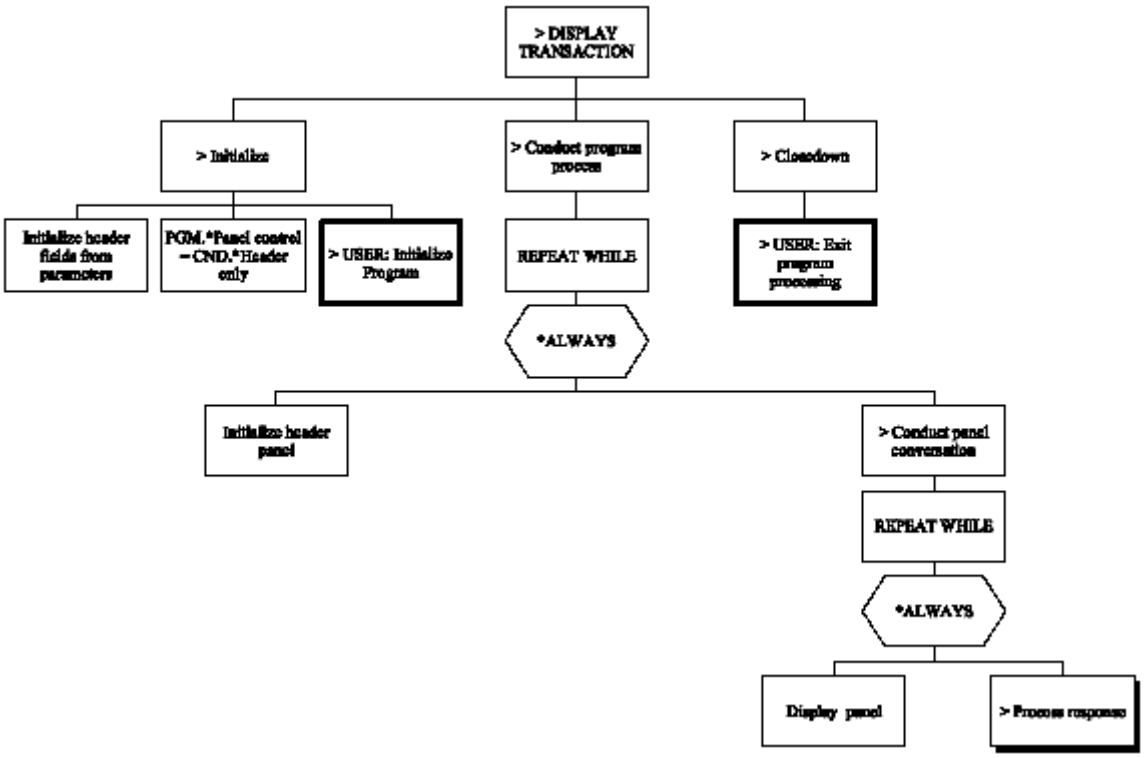
Display Record – 3 Panels (Chart 7 of 8)



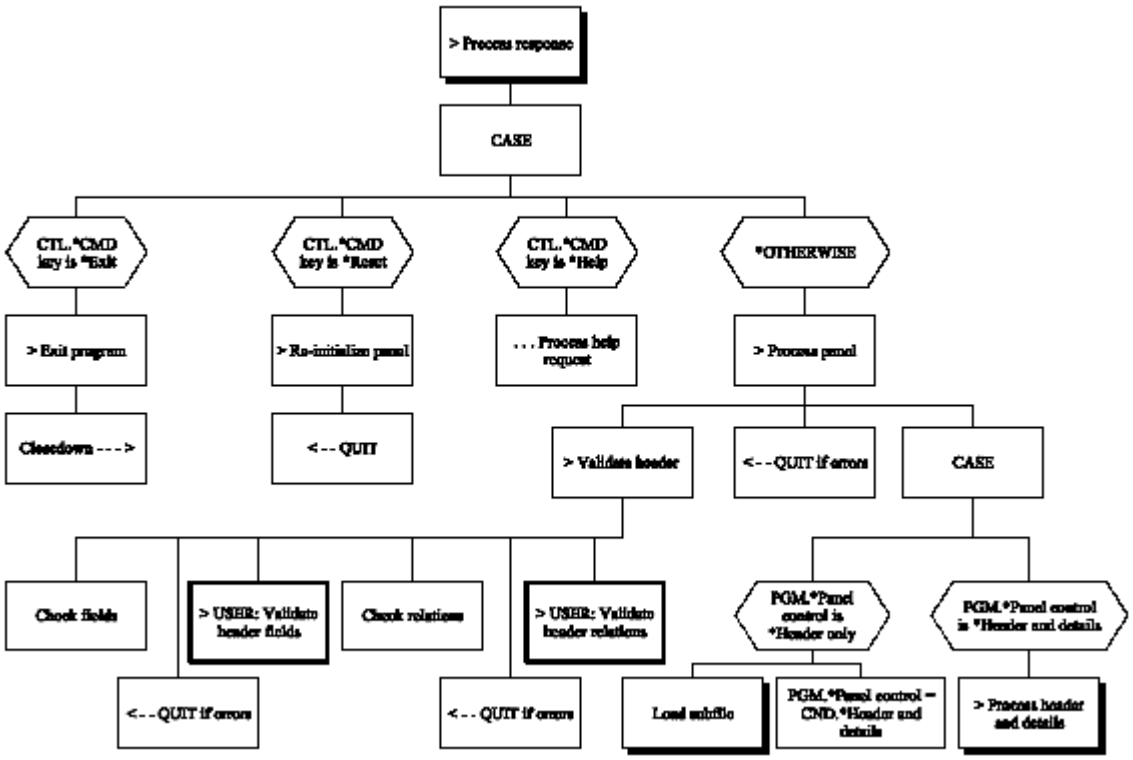
Display Record – 3 Panels (Chart 8 of 8)



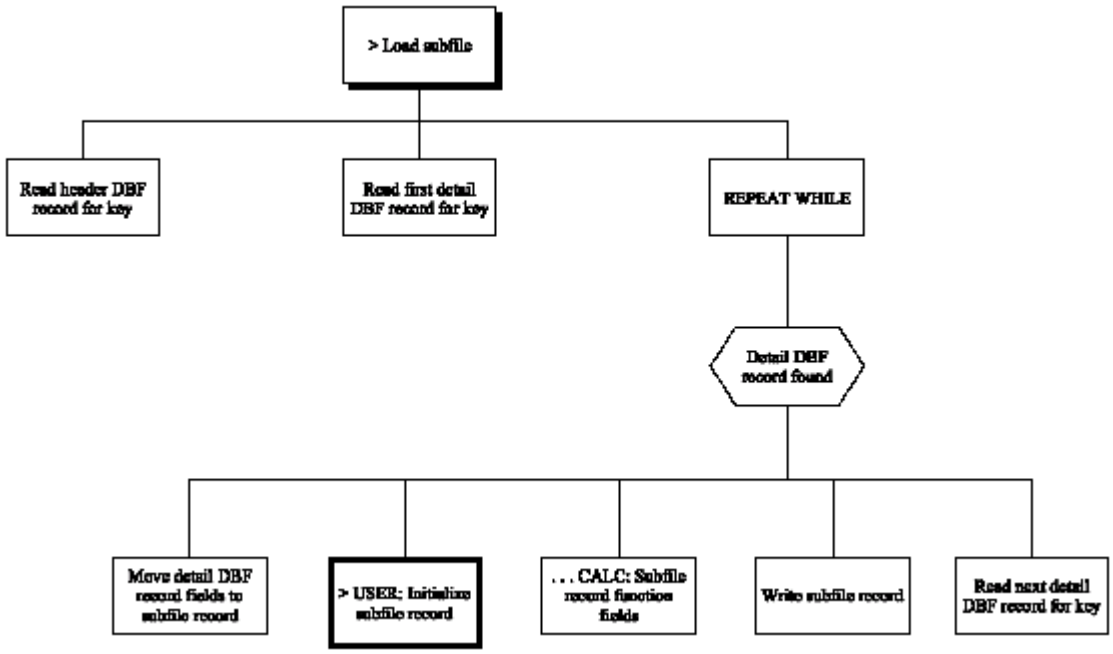
Display Transaction (Chart 1 of 6)



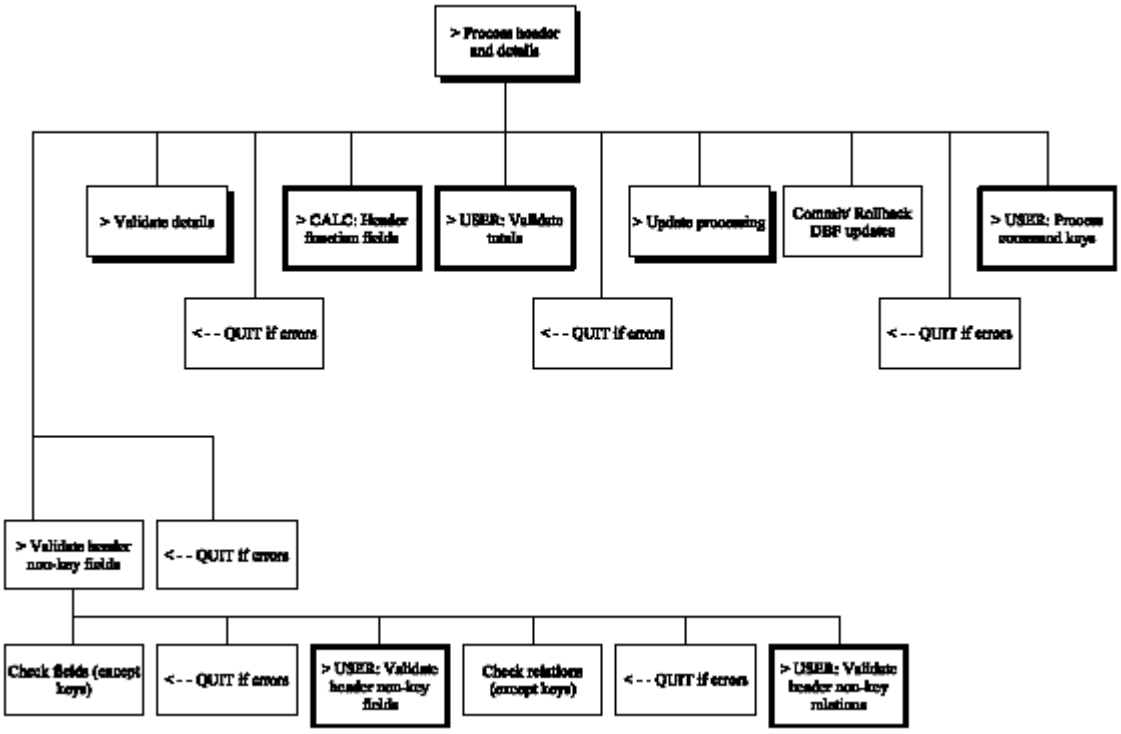
Display Transaction (Chart 2 of 6)



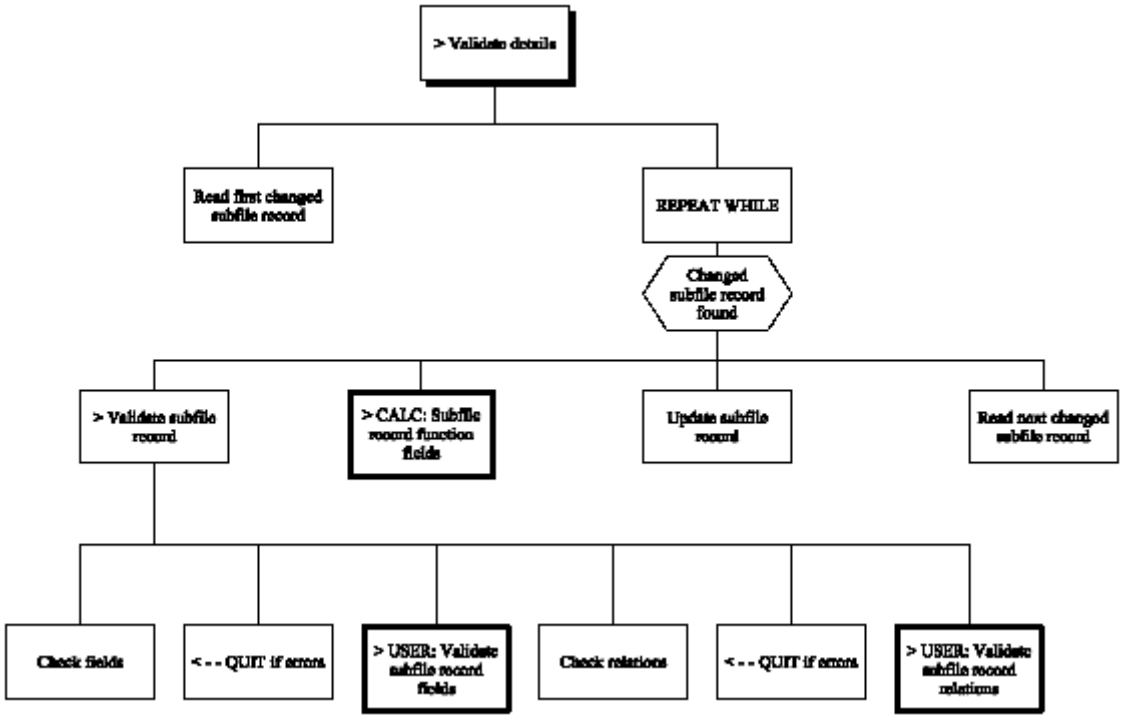
Display Transaction (Chart 3 of 6)



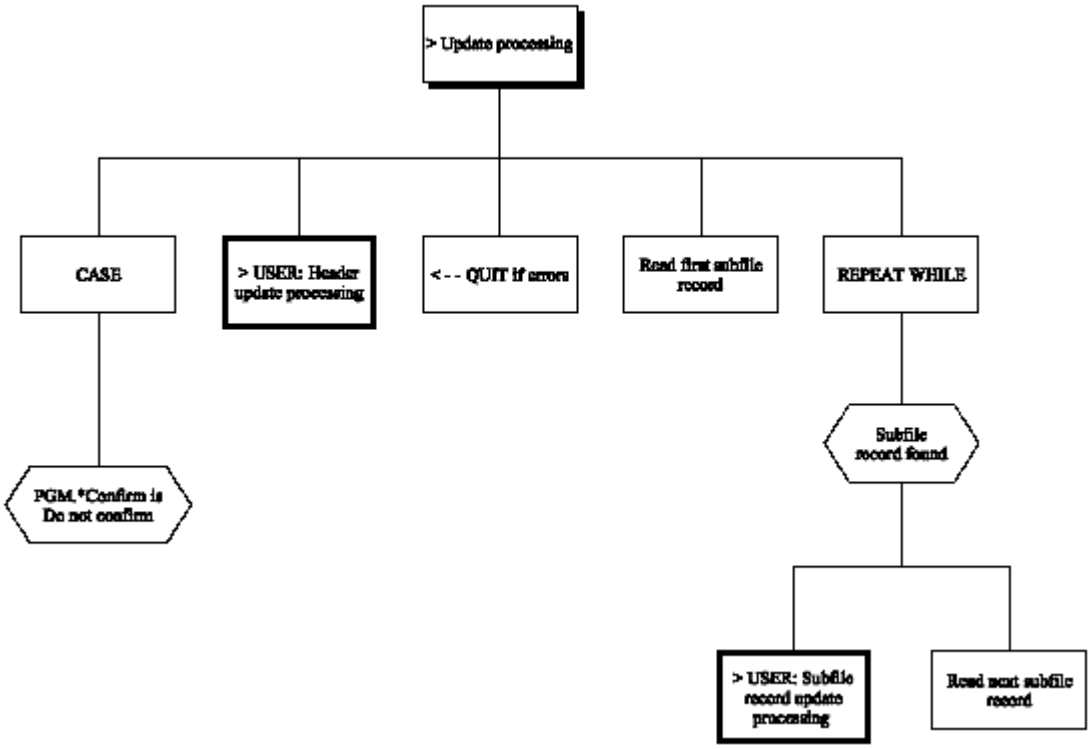
Display Transaction (Chart 4 of 6)



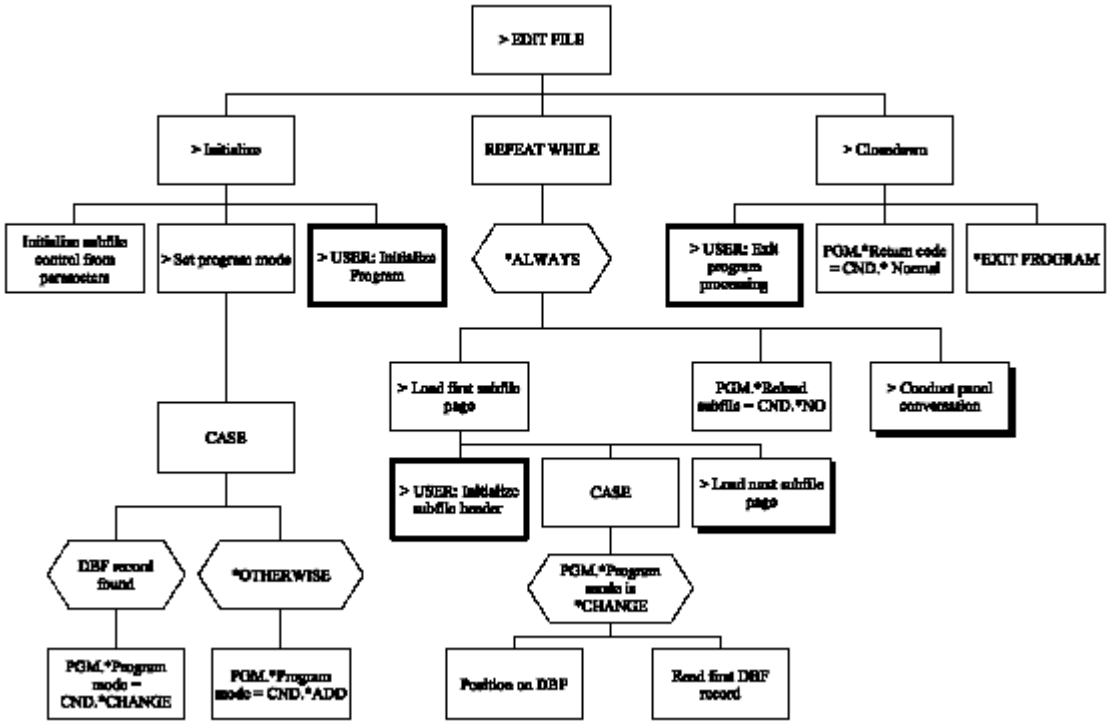
Display Transaction (Chart 5 of 6)



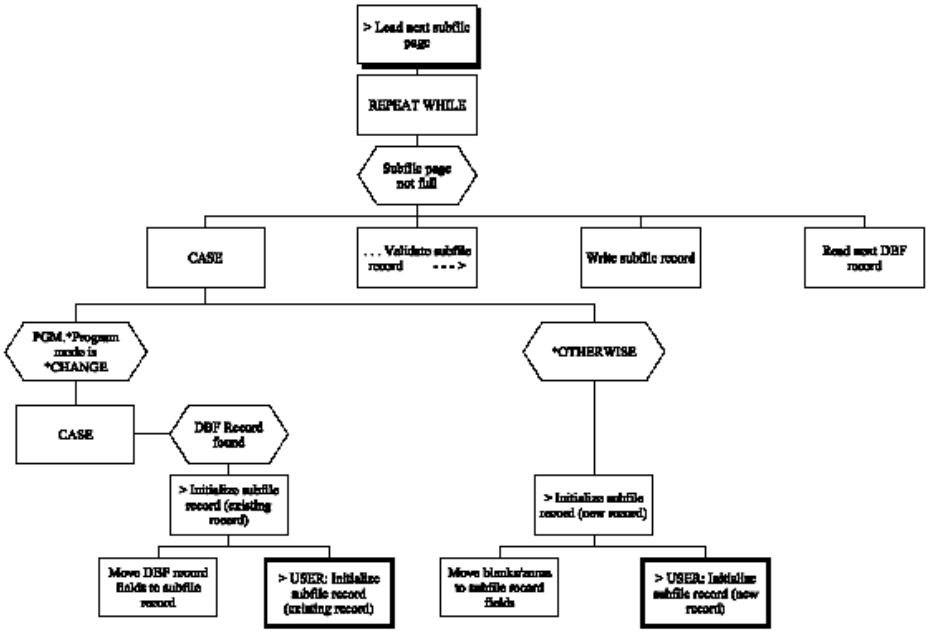
Display Transaction (Chart 6 of 6)



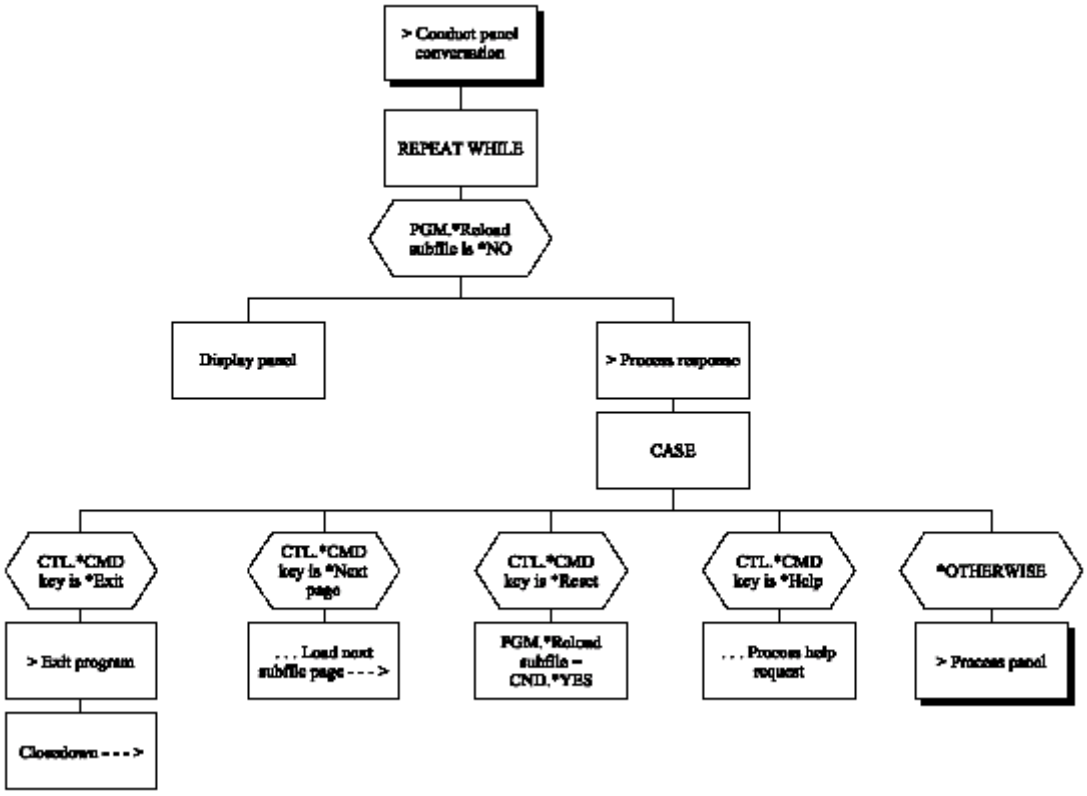
Edit File (Chart 1 of 7)



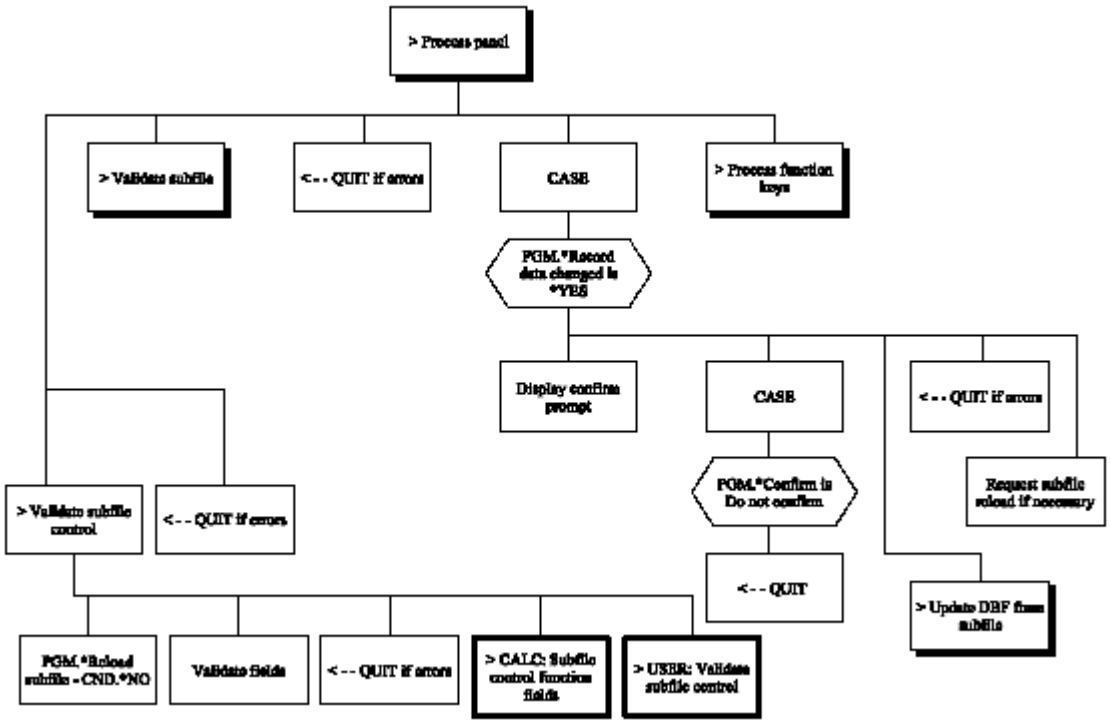
Edit File (Chart 2 of 7)



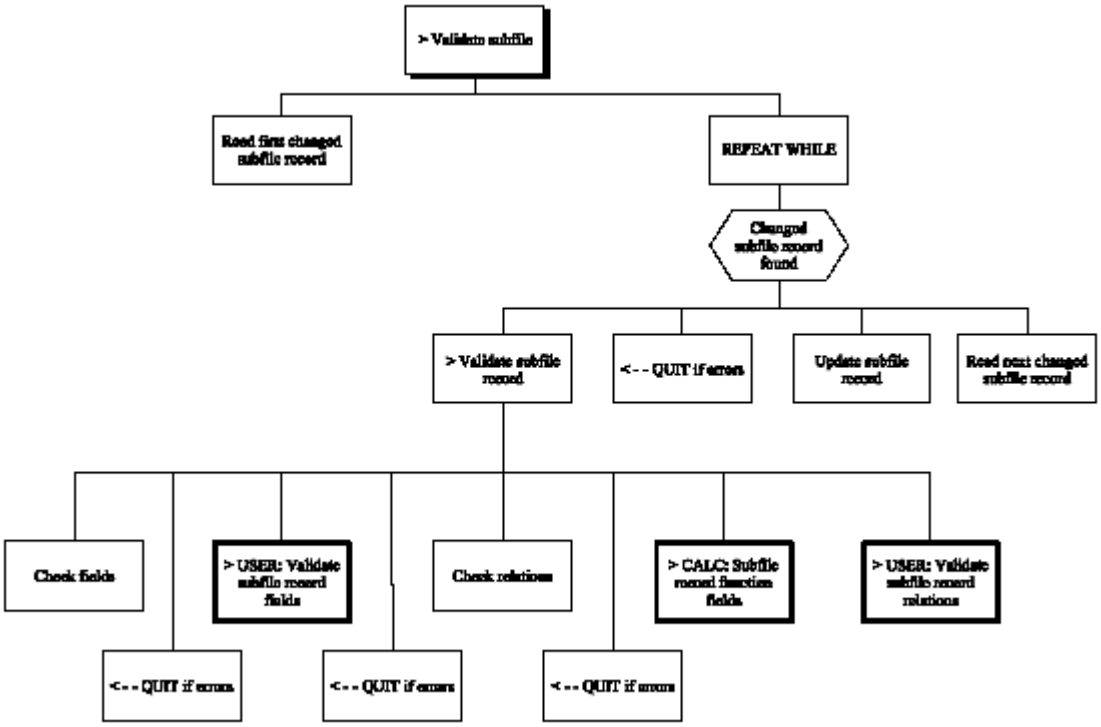
Edit File (Chart 3 of 7)



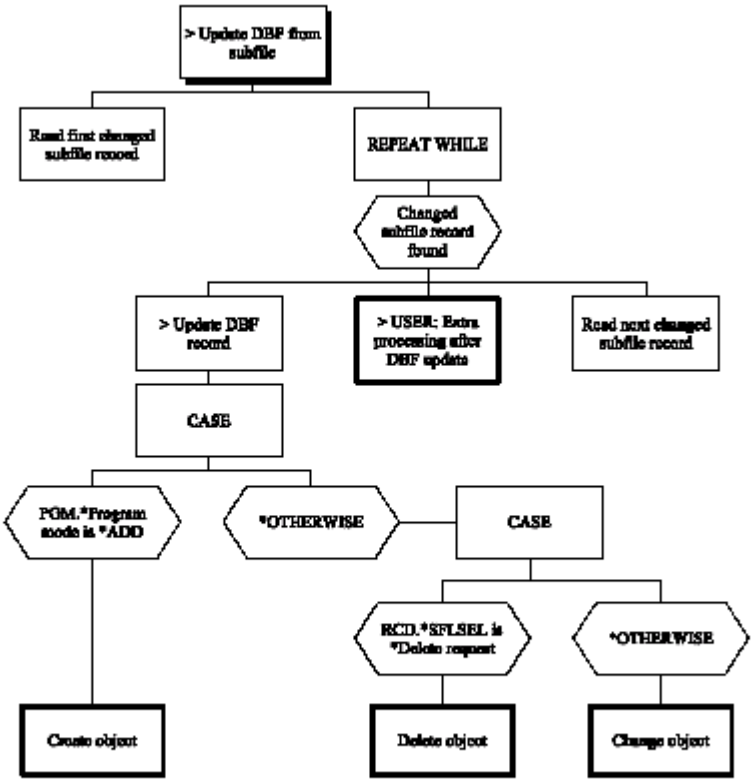
Edit File (Chart 4 of 7)



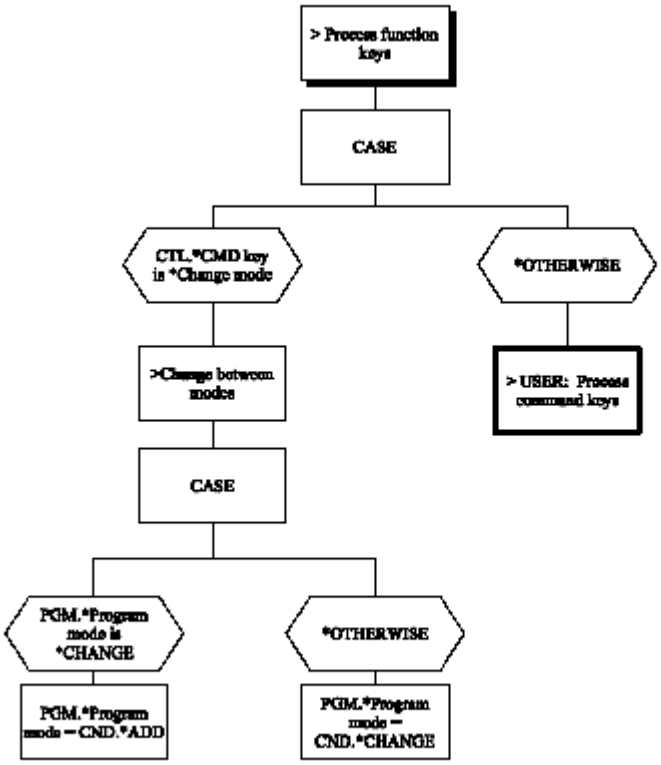
Edit File (Chart 5 of 7)



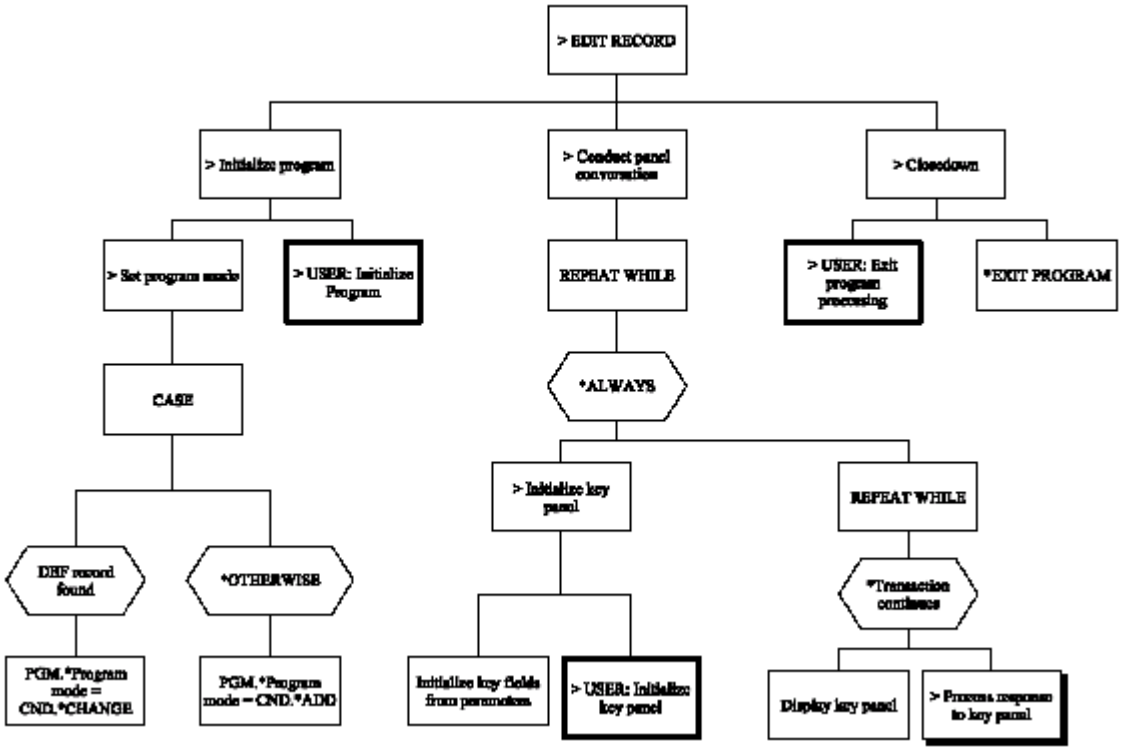
Edit File (Chart 6 of 7)



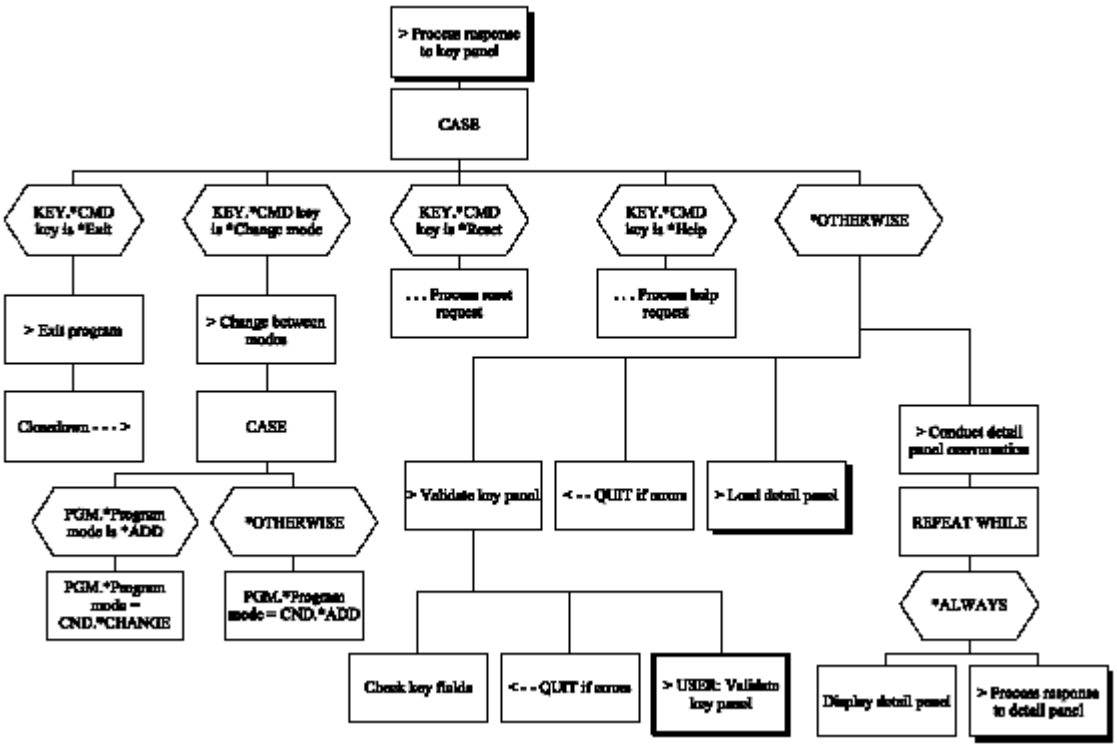
Edit File (Chart 7 of 7)



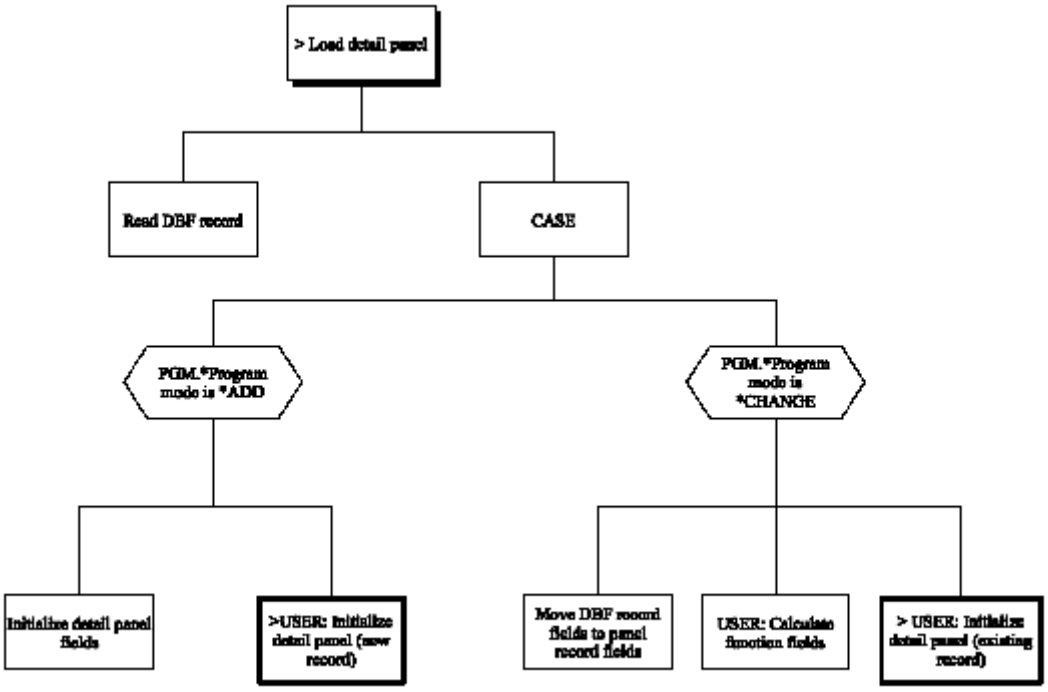
Edit Record (Chart 1 of 5)



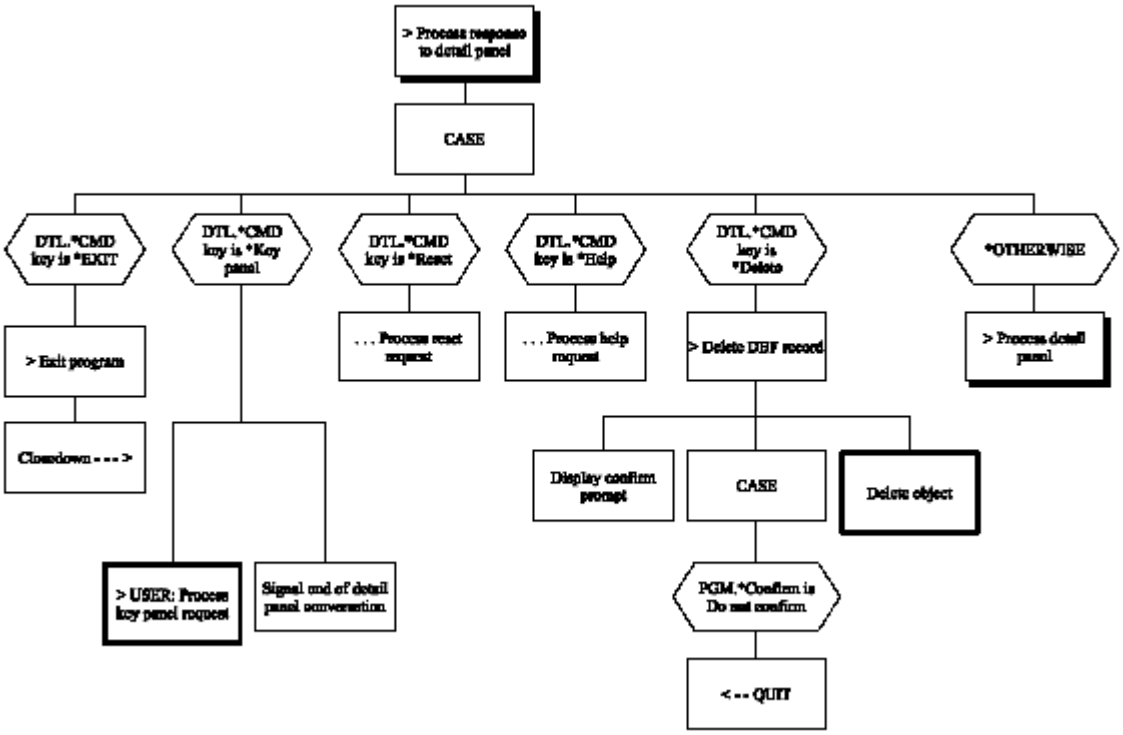
Edit Record (Chart 2 of 5)



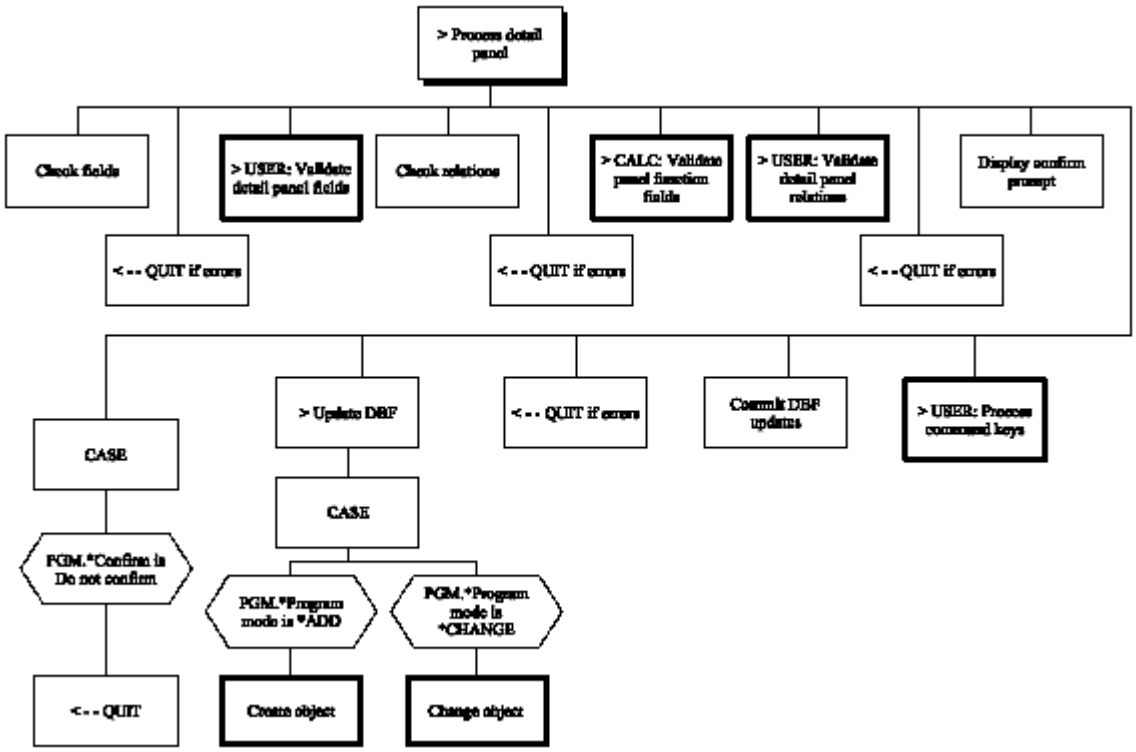
Edit Record (Chart 3 of 5)



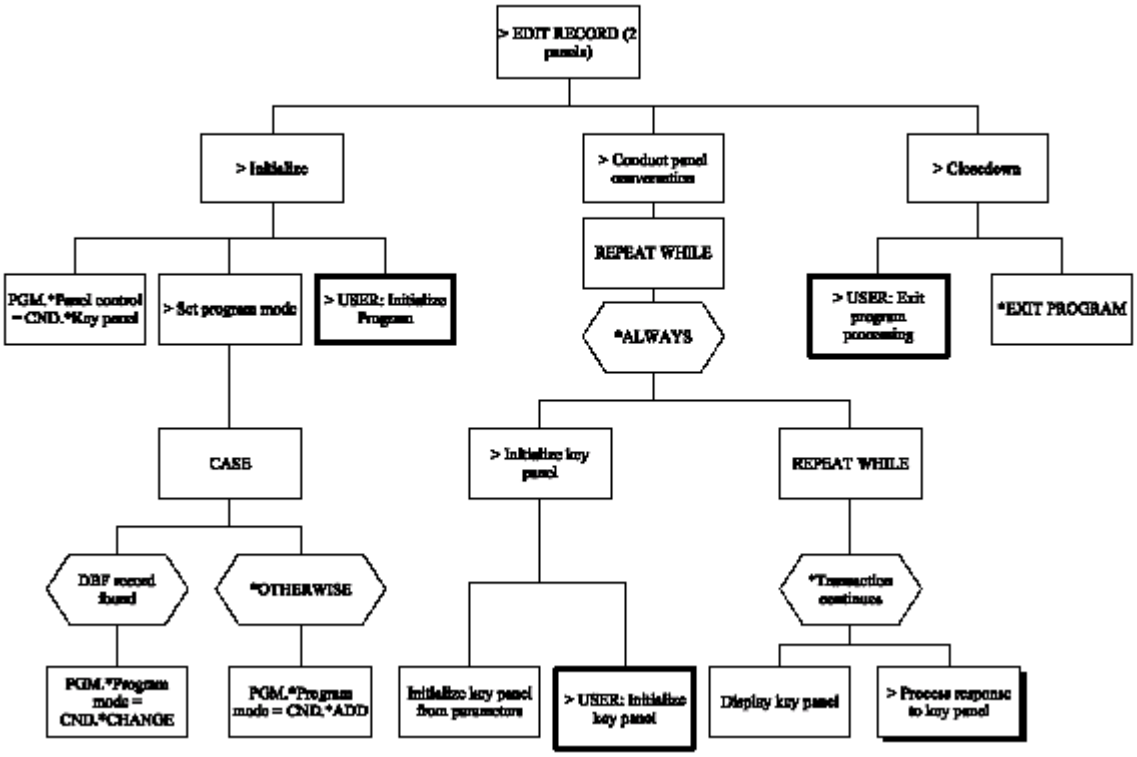
Edit Record (Chart 4 of 5)



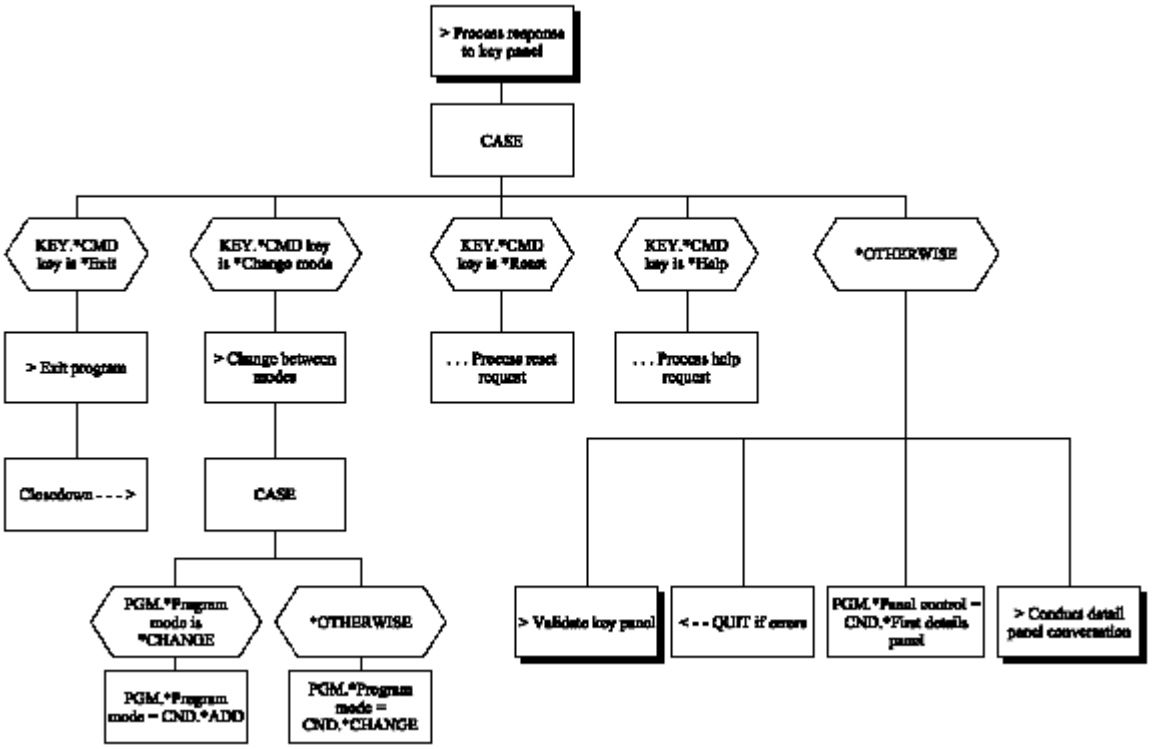
Edit Record (Chart 5 of 5)



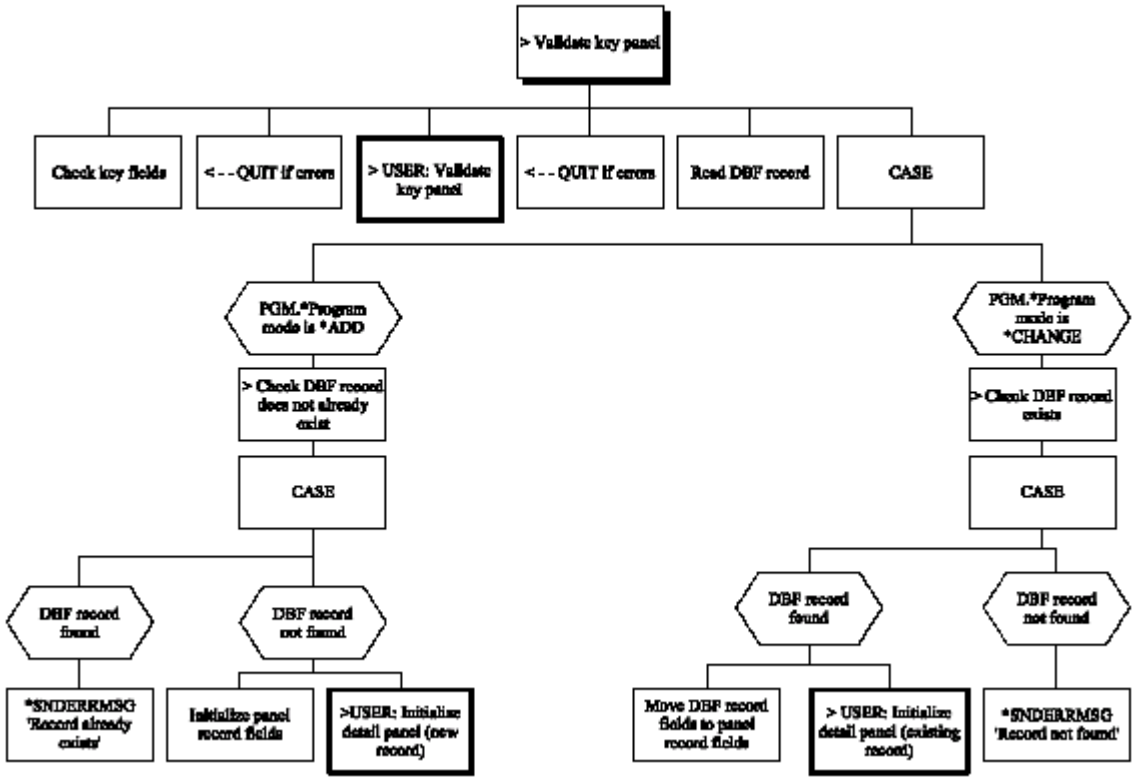
Edit Record – 2 Panels (Chart 1 of 9)



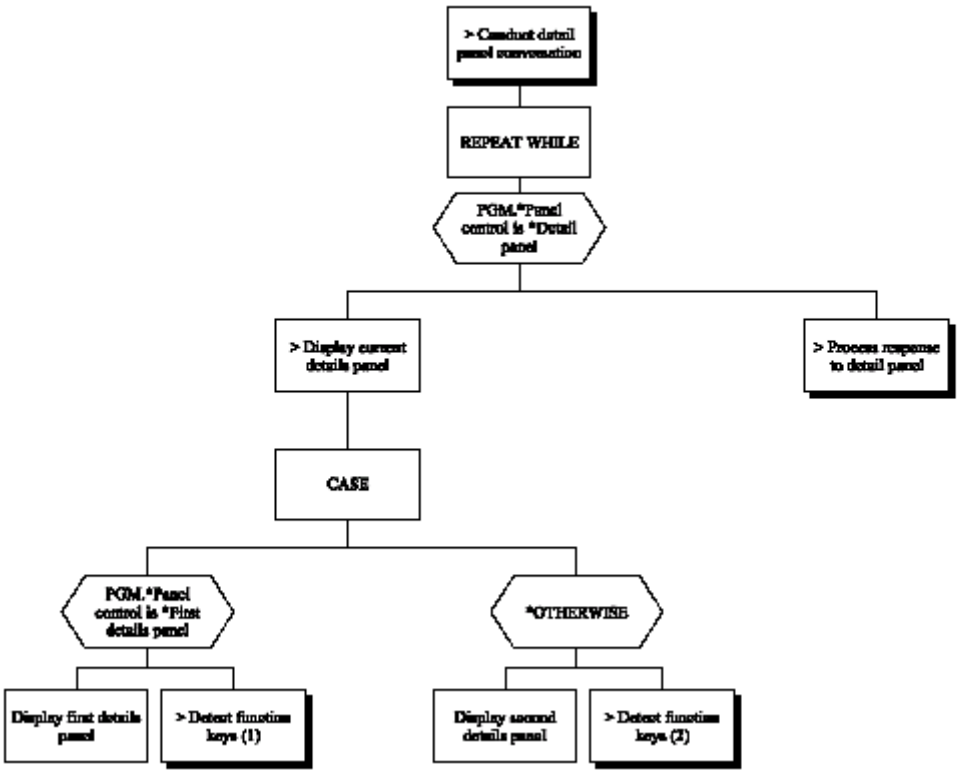
Edit Record – 2 Panels (Chart 2 of 9)



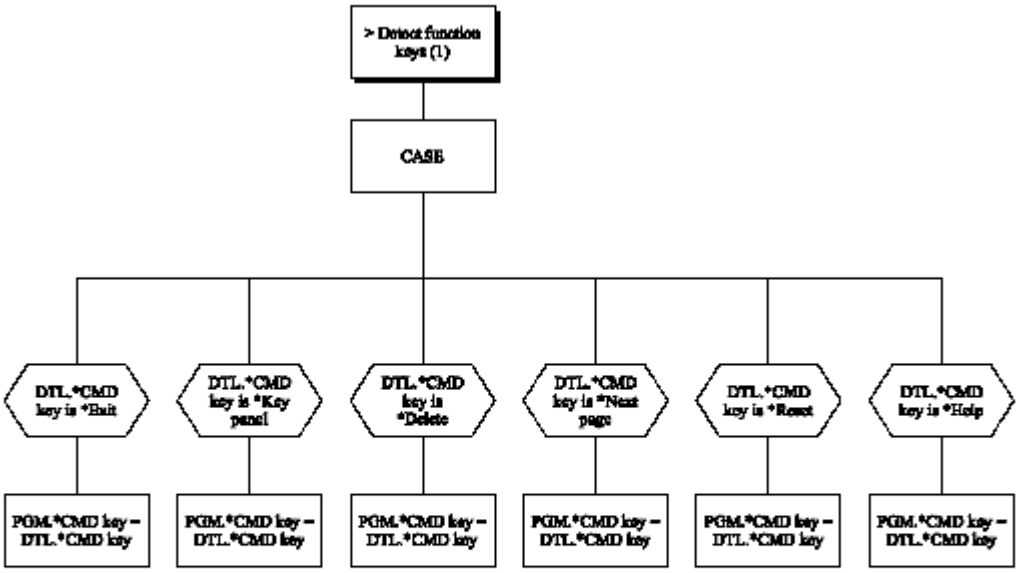
Edit Record – 2 Panels (Chart 3 of 9)



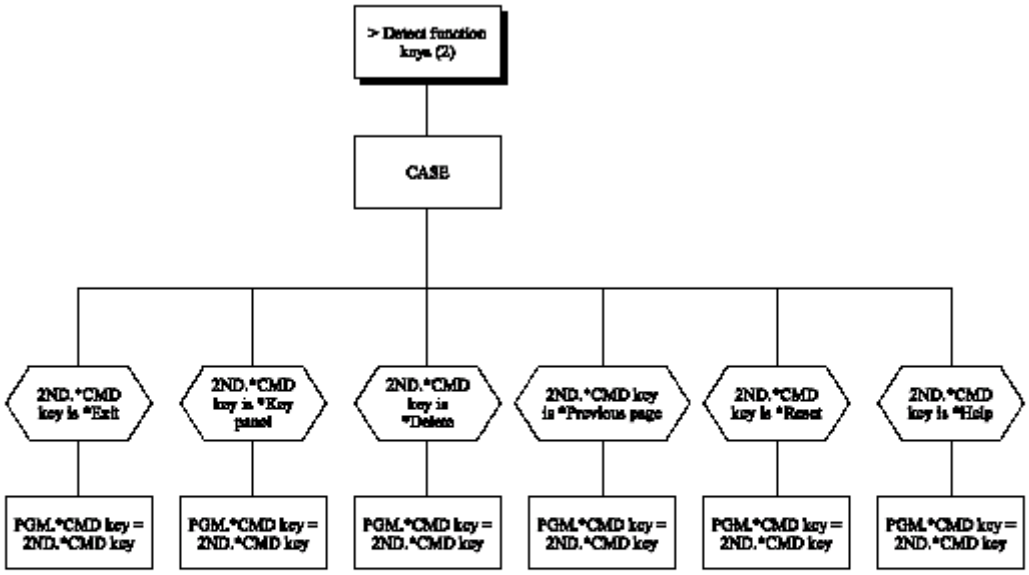
Edit Record – 2 Panels (Chart 4 of 9)



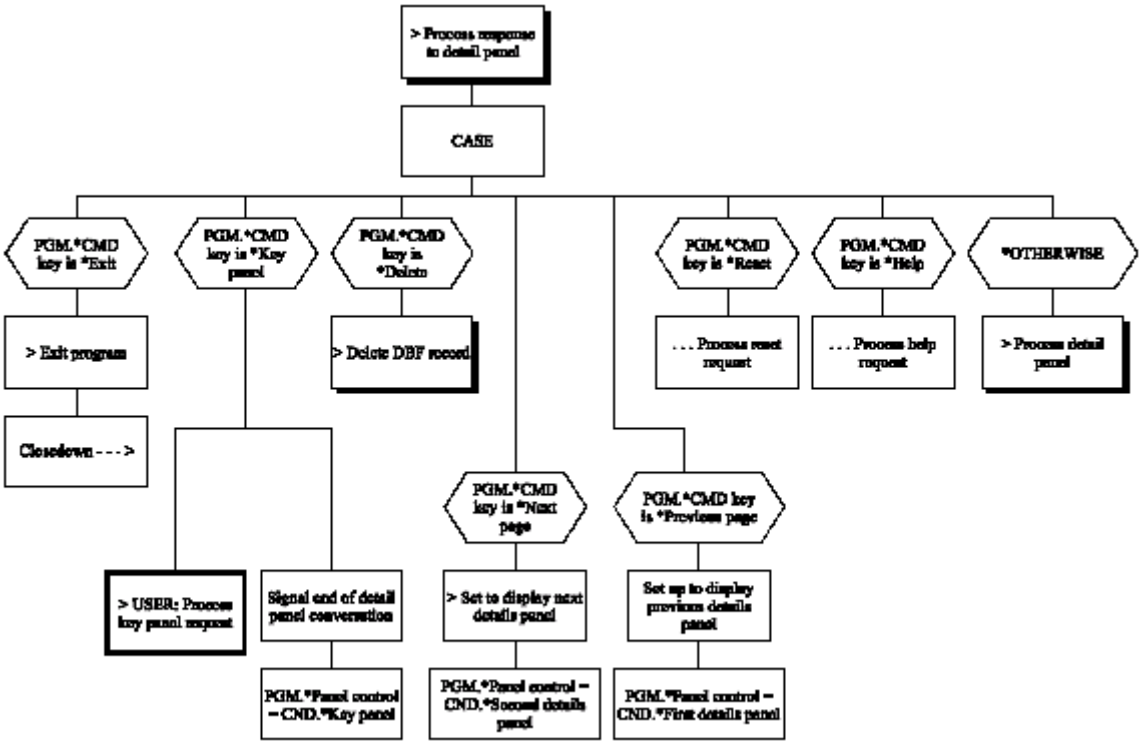
Edit Record – 2 Panels (Chart 5 of 9)



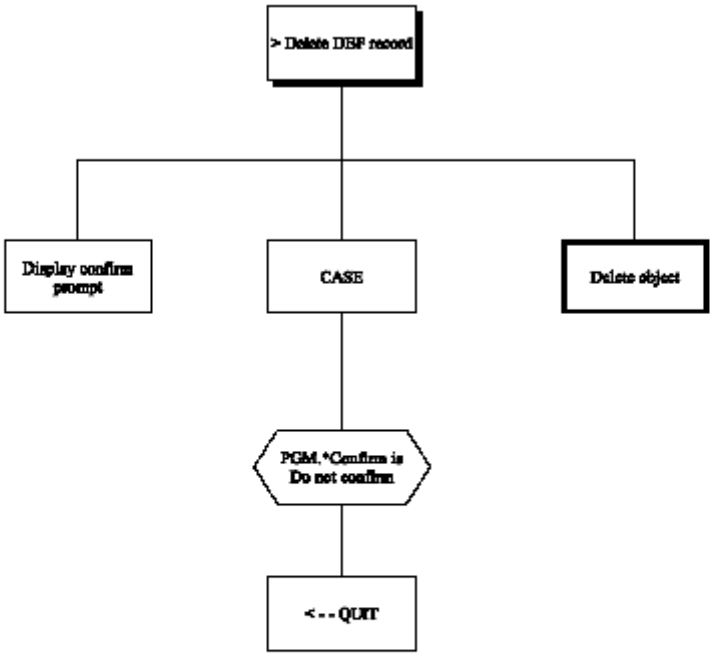
Edit Record – 2 Panels (Chart 6 of 9)



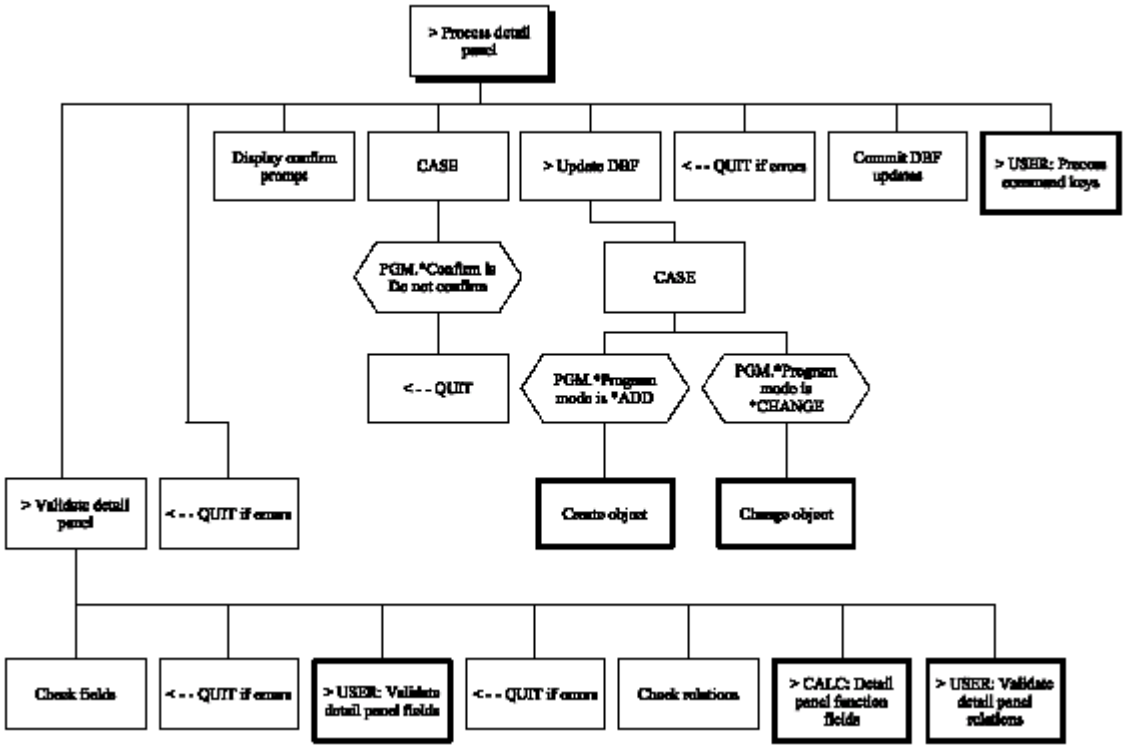
Edit Record – 2 Panels (Chart 7 of 9)



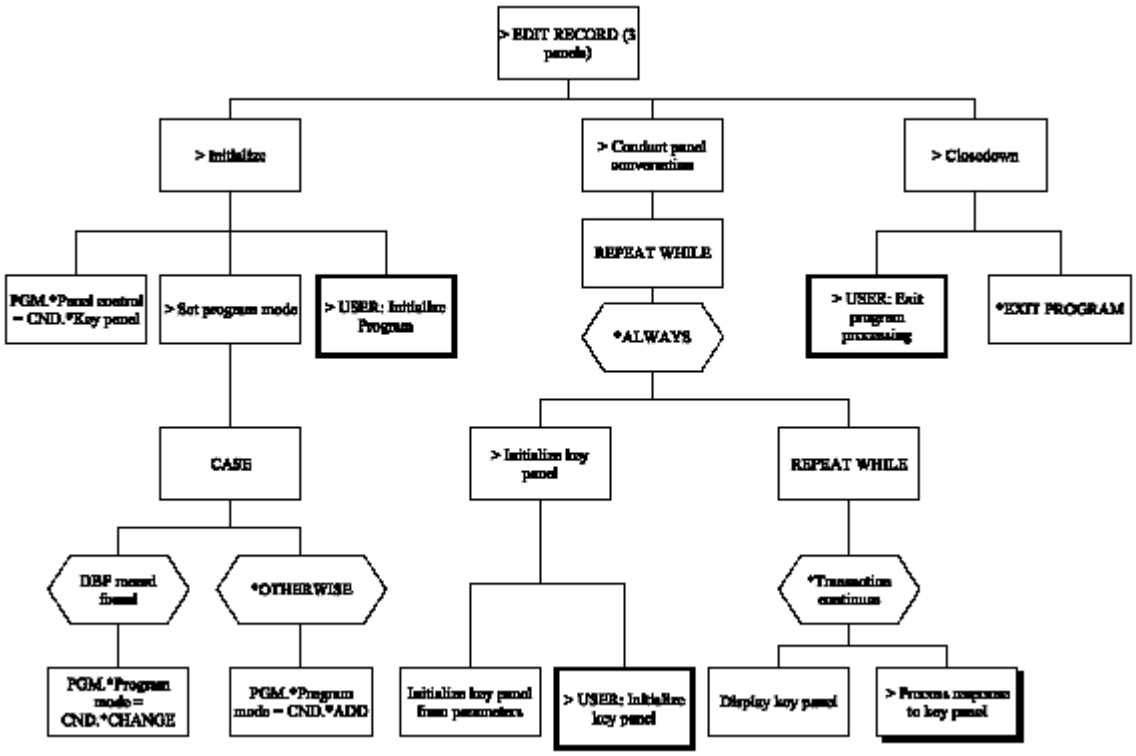
Edit Record – 2 Panels (Chart 8 of 9)



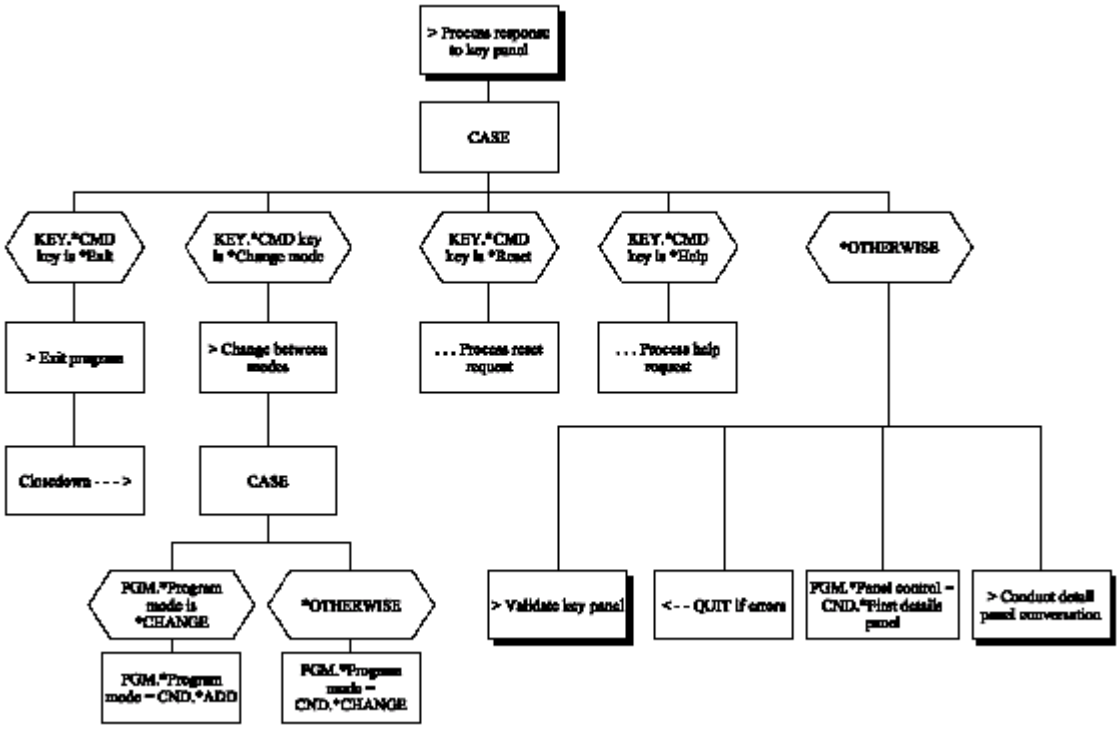
Edit Record – 2 Panels (Chart 9 of 9)



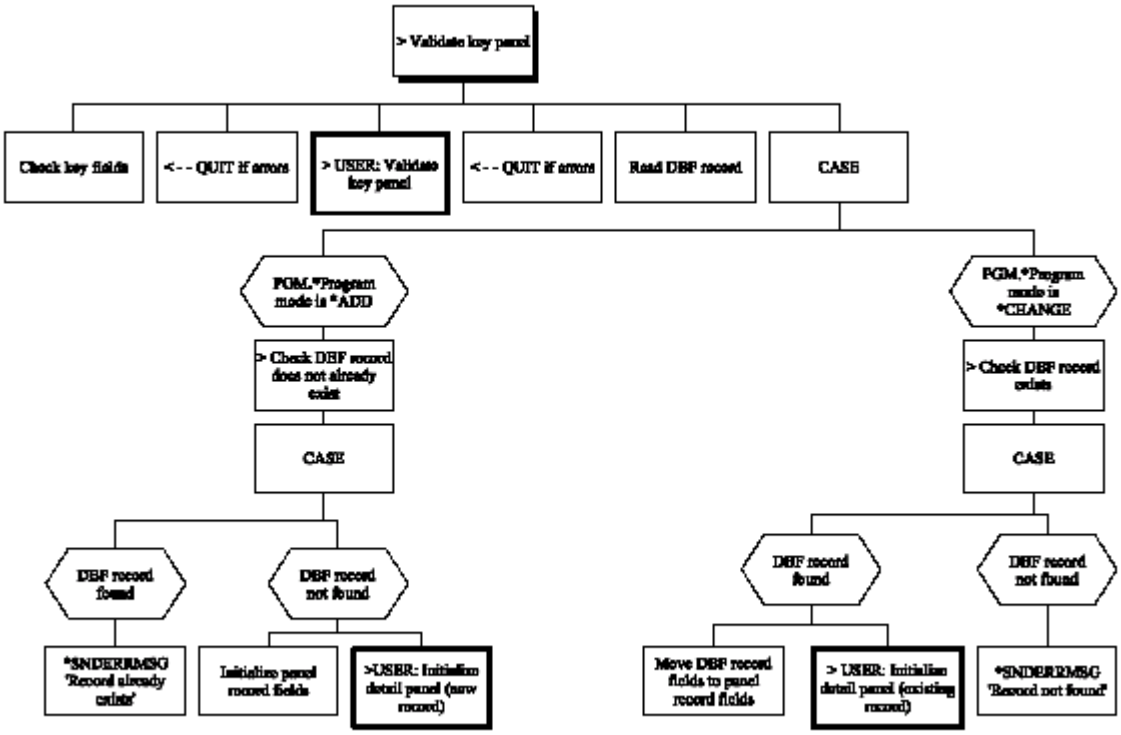
Edit Record – 3 Panels (Chart 1 of 10)



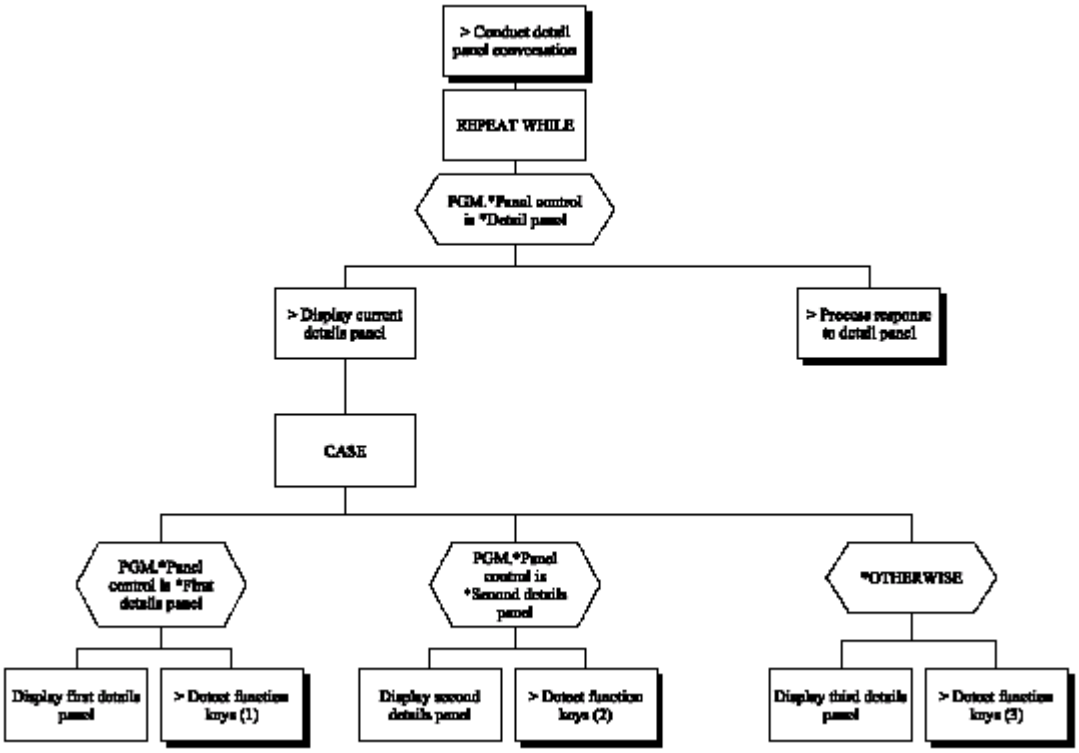
Edit Record – 3 Panels (Chart 2 of 10)



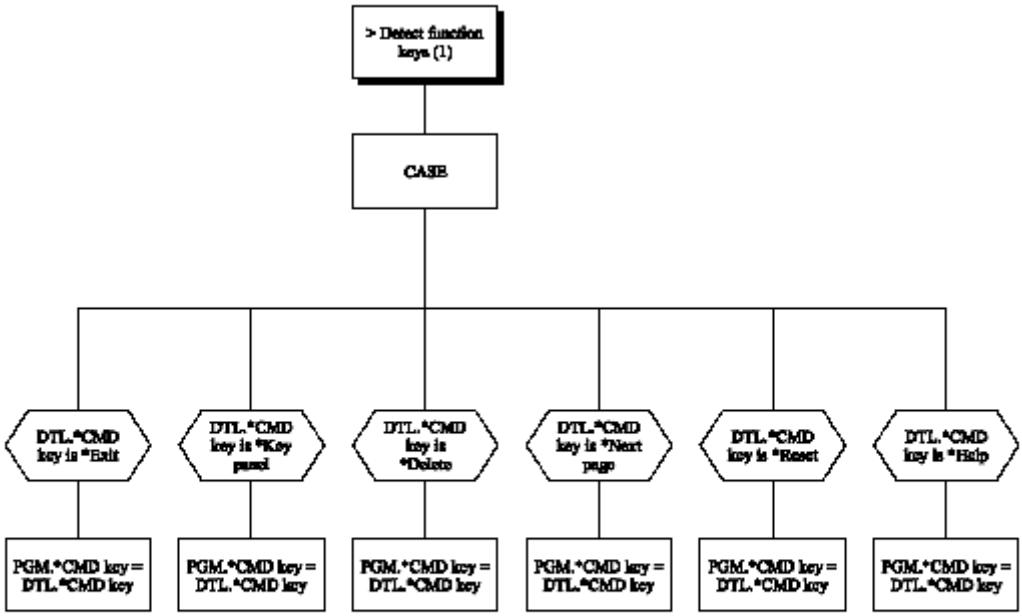
Edit Record – 3 Panels (Chart 3 of 10)



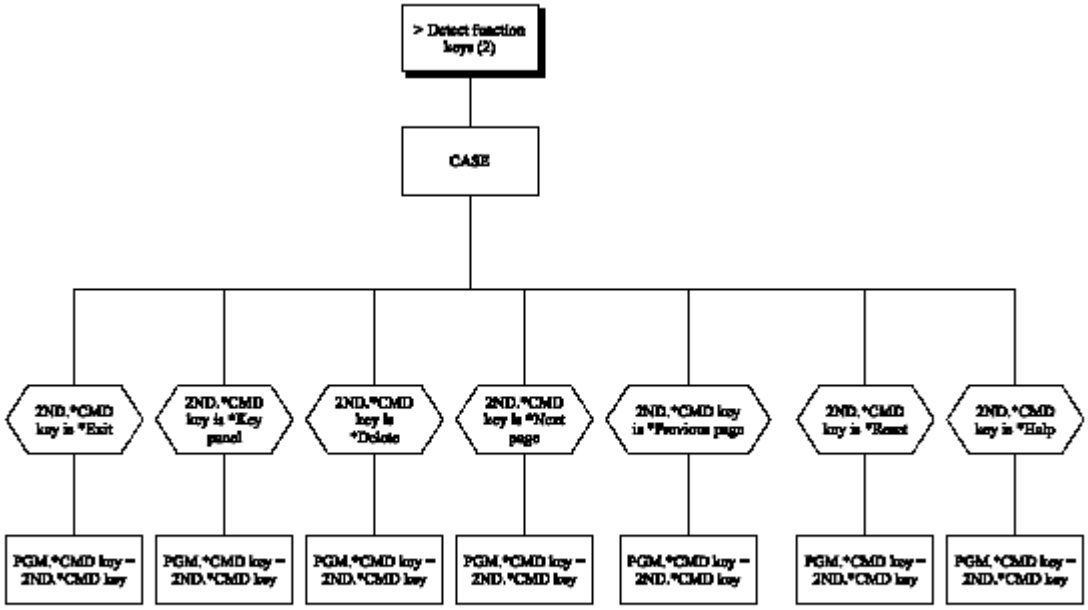
Edit Record – 3 Panels (Chart 4 of 10)



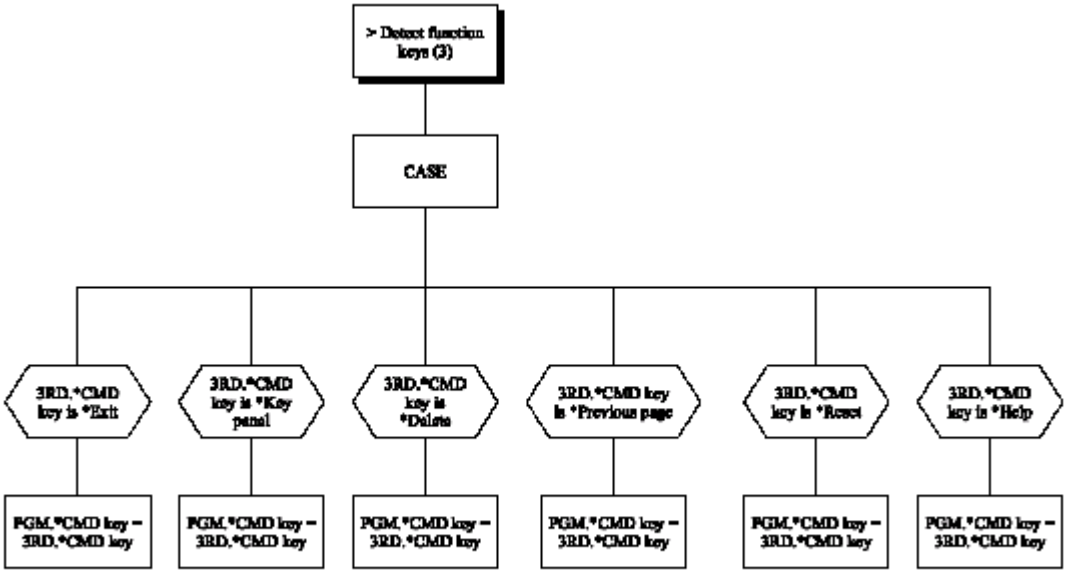
Edit Record – 3 Panels (Chart 5 of 10)



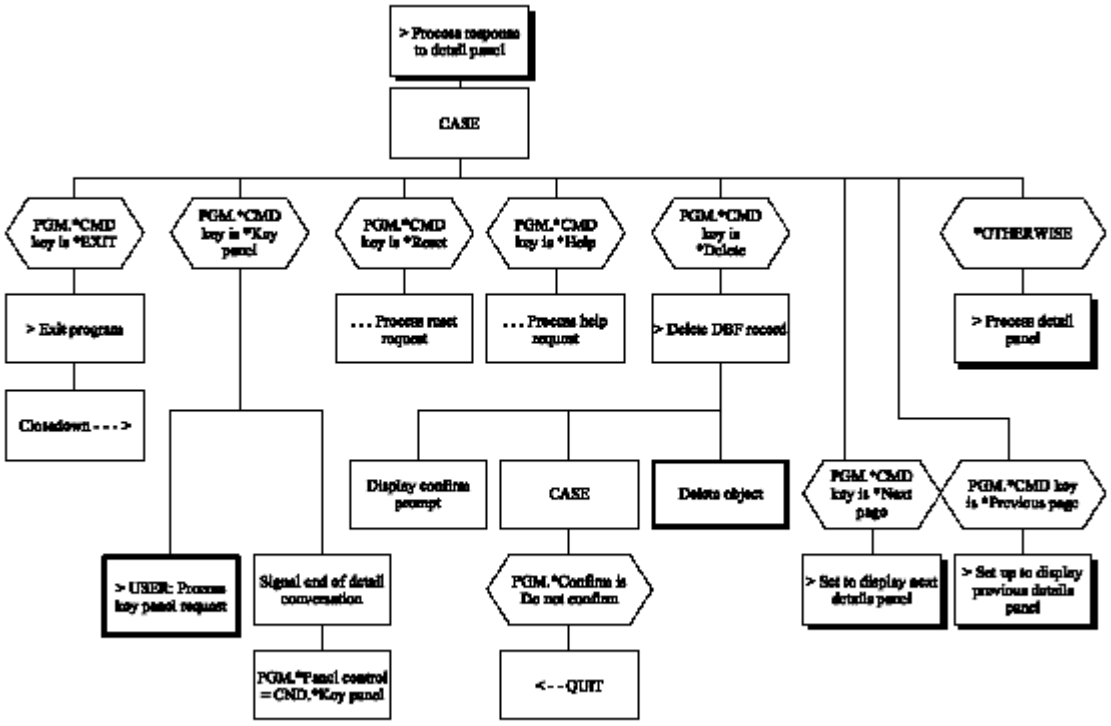
Edit Record – 3 Panels (Chart 6 of 10)



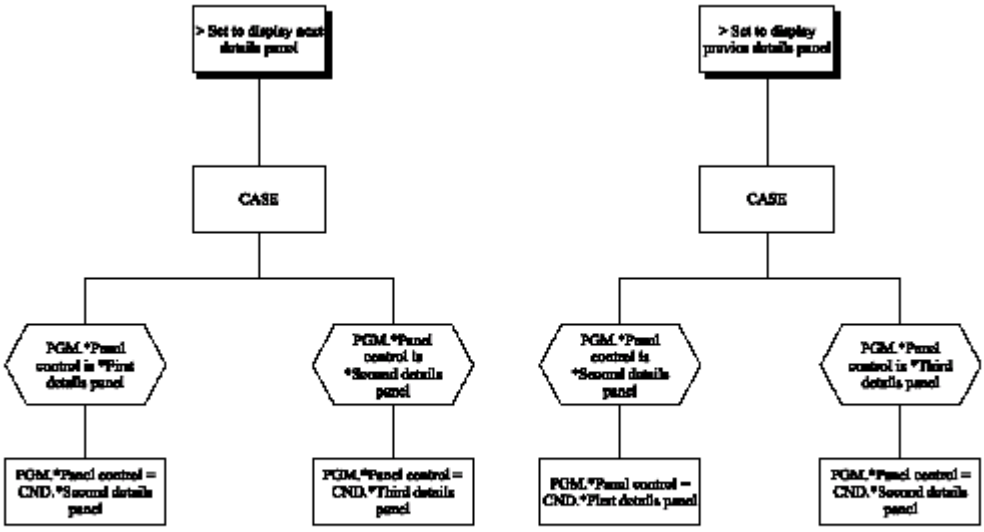
Edit Record – 3 Panels (Chart 7 of 10)



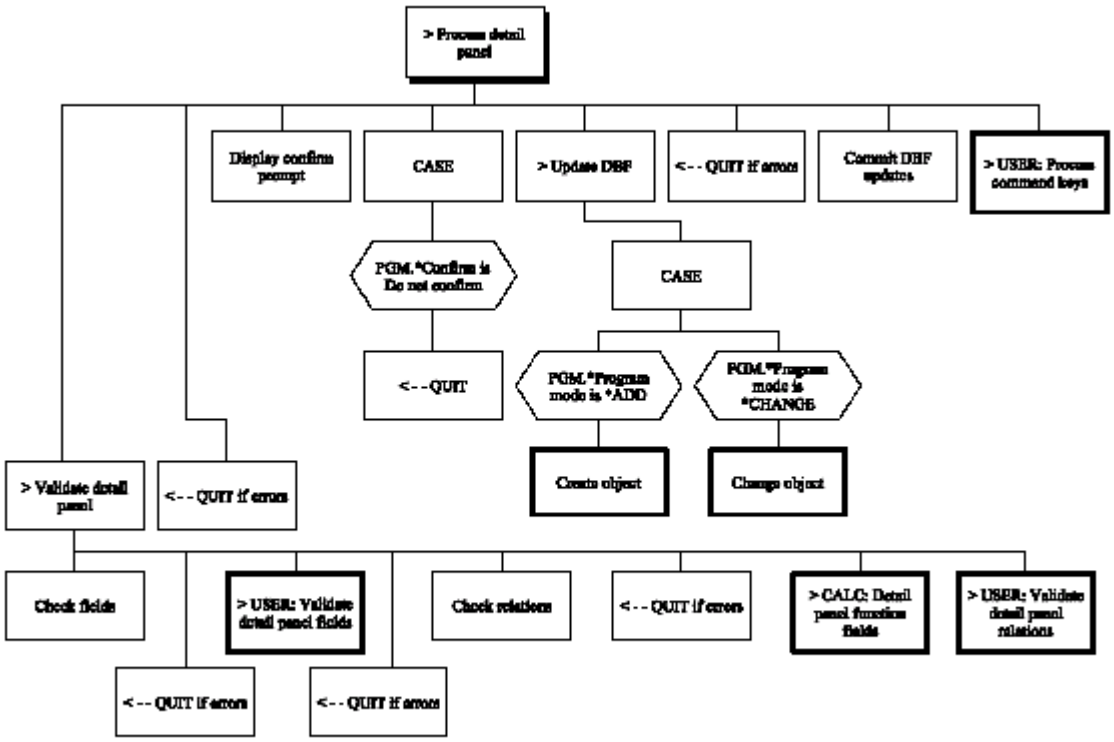
Edit Record – 3 Panels (Chart 8 of 10)



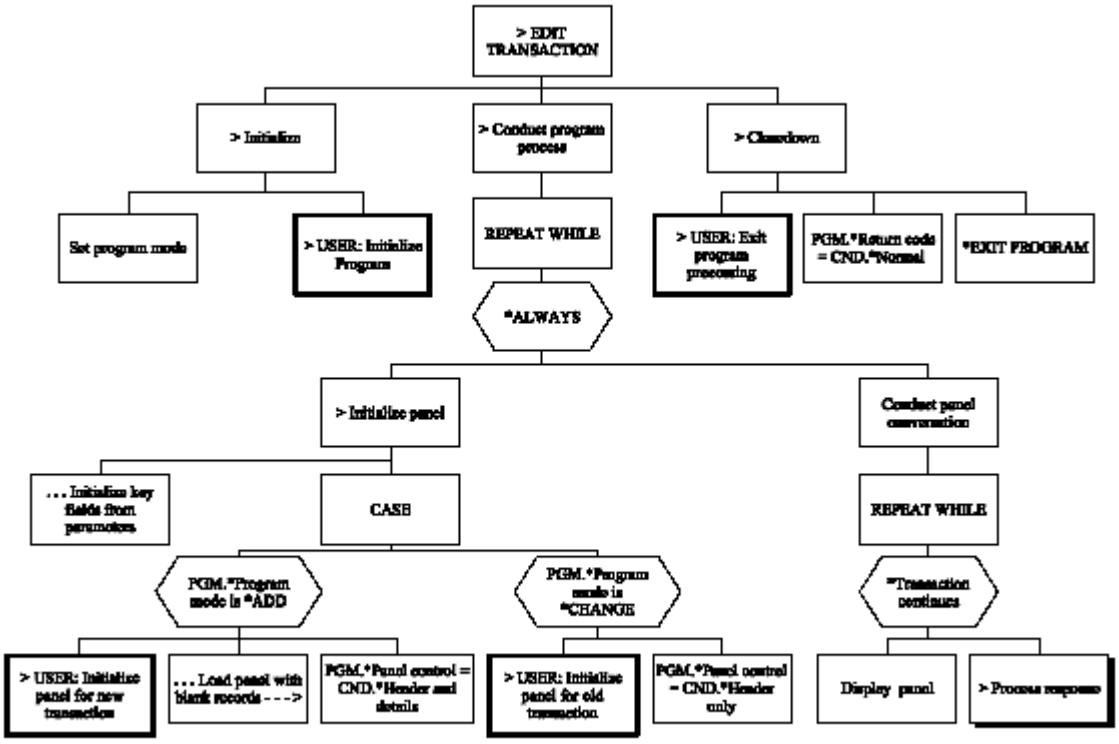
Edit Record – 3 Panels (Chart 9 of 10)



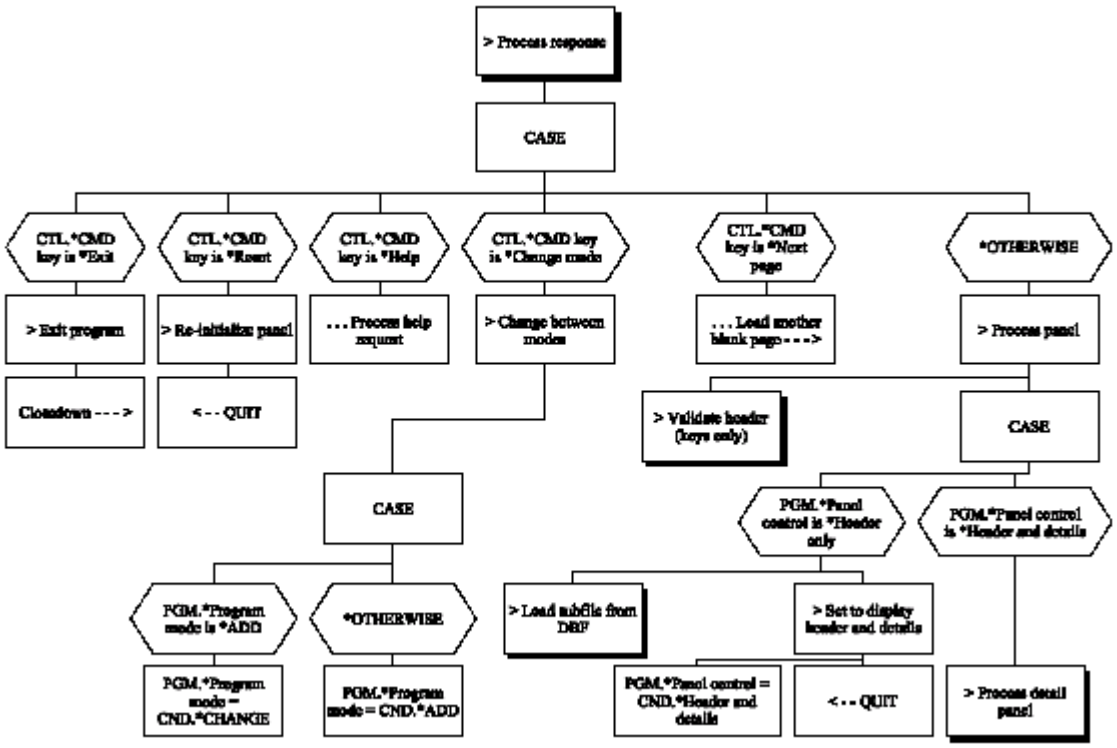
Edit Record – 3 Panels (Chart 10 of 10)



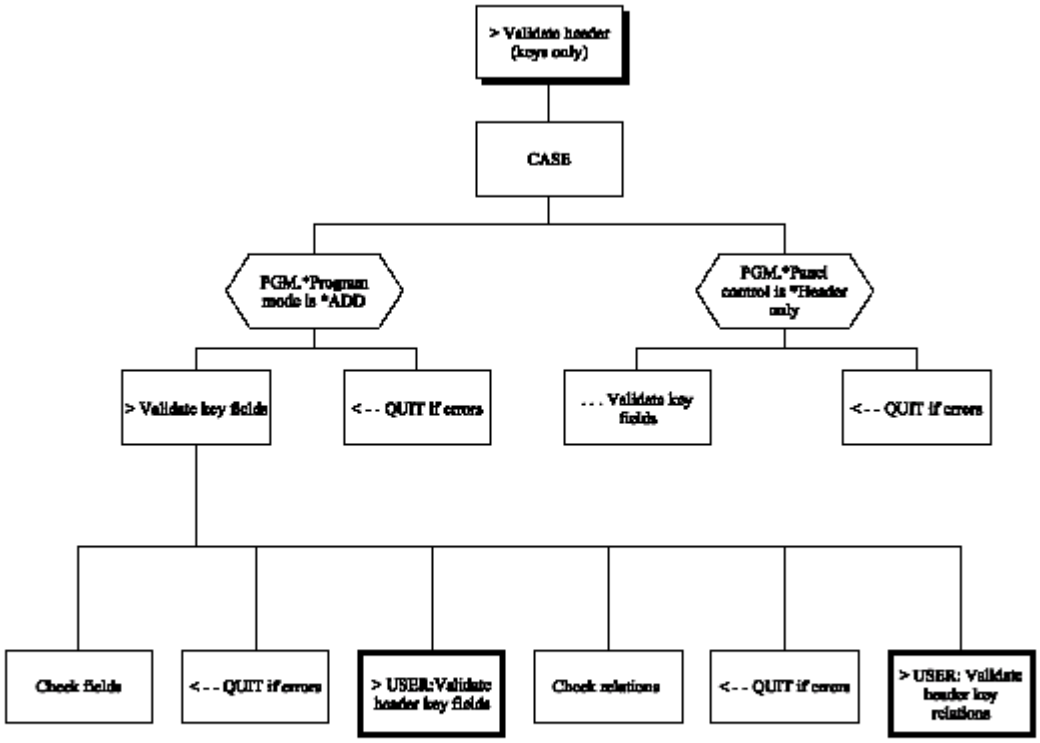
Edit Transaction (Chart 1 of 8)



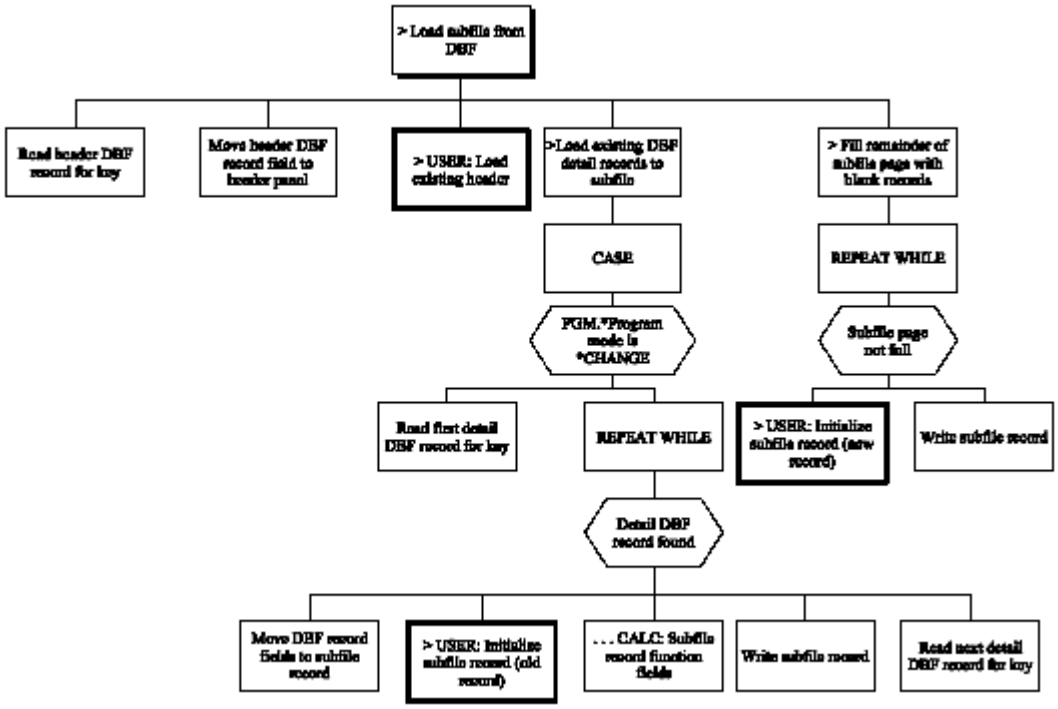
Edit Transaction (Chart 2 of 8)



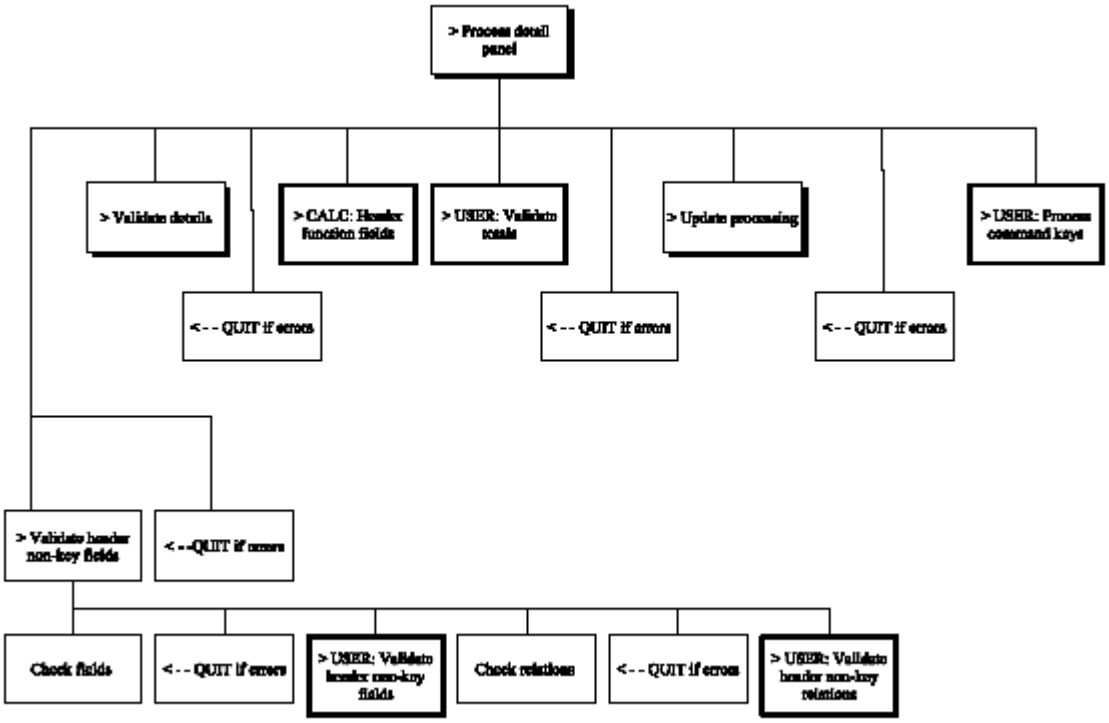
Edit Transaction (Chart 3 of 8)



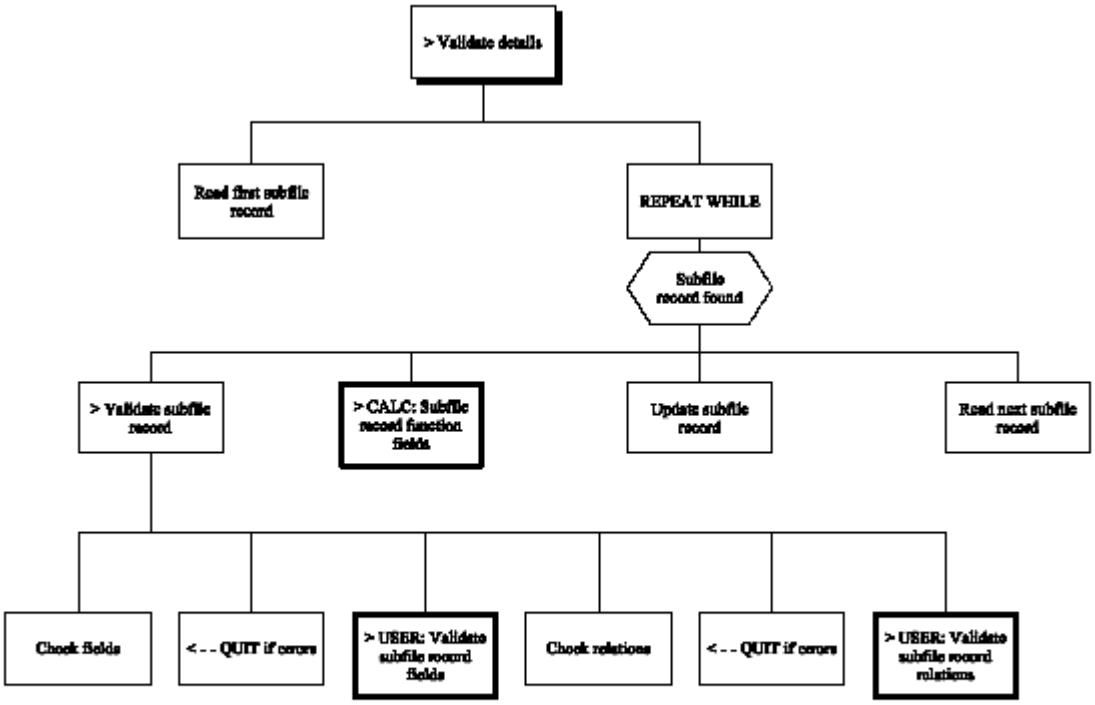
Edit Transaction (Chart 4 of 8)



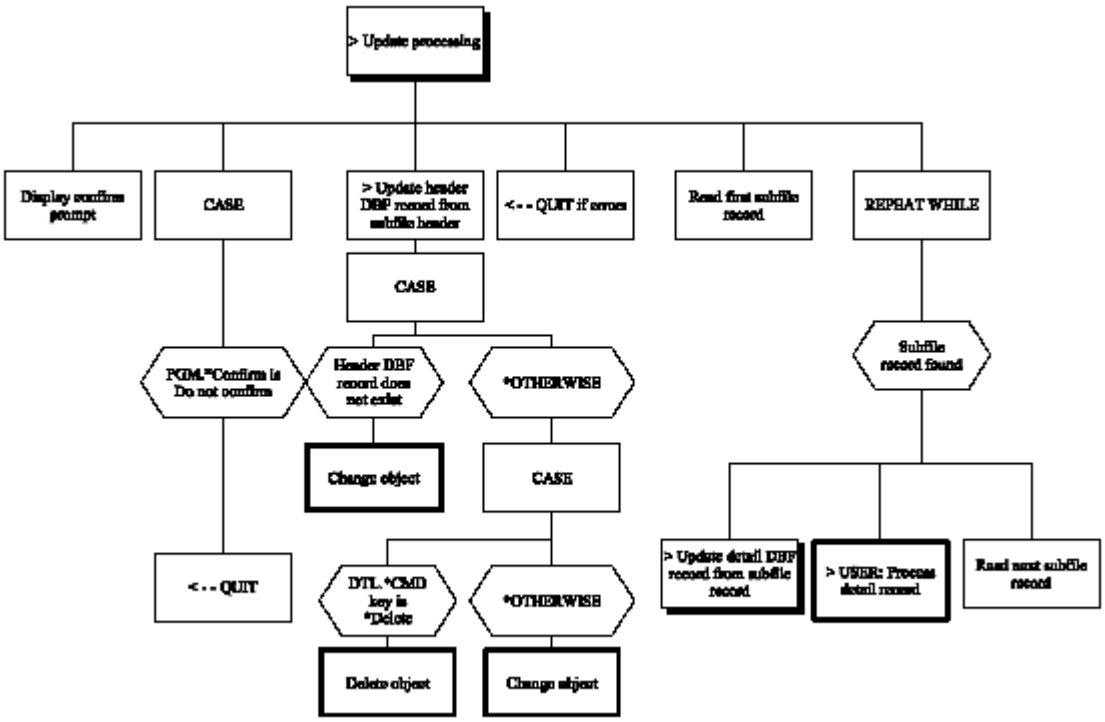
Edit Transaction (Chart 5 of 8)



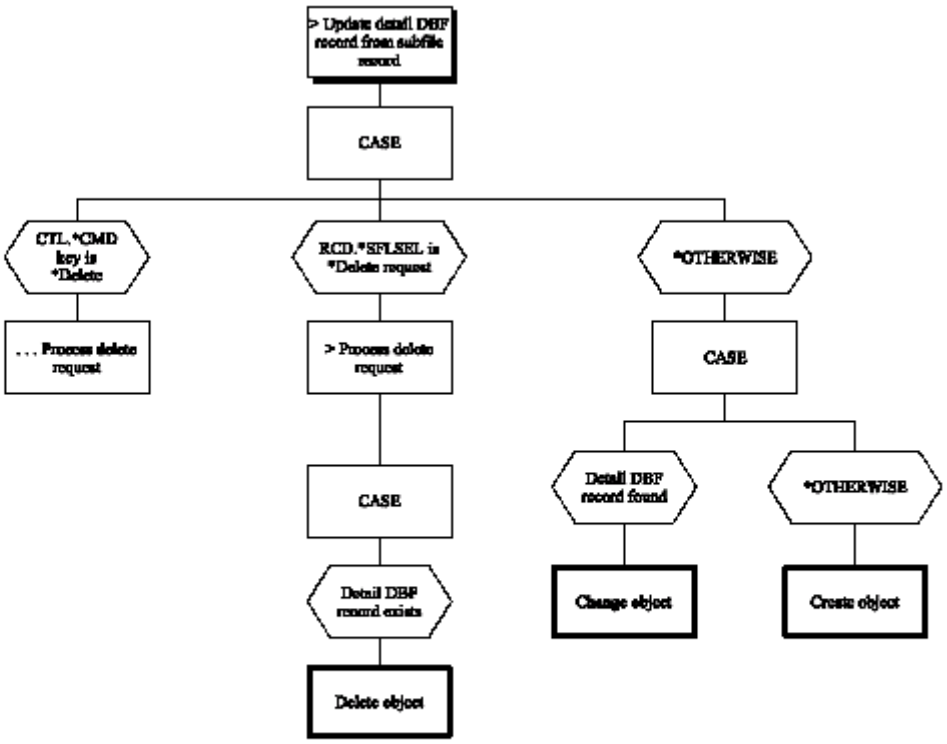
Edit Transaction (Chart 6 of 8)



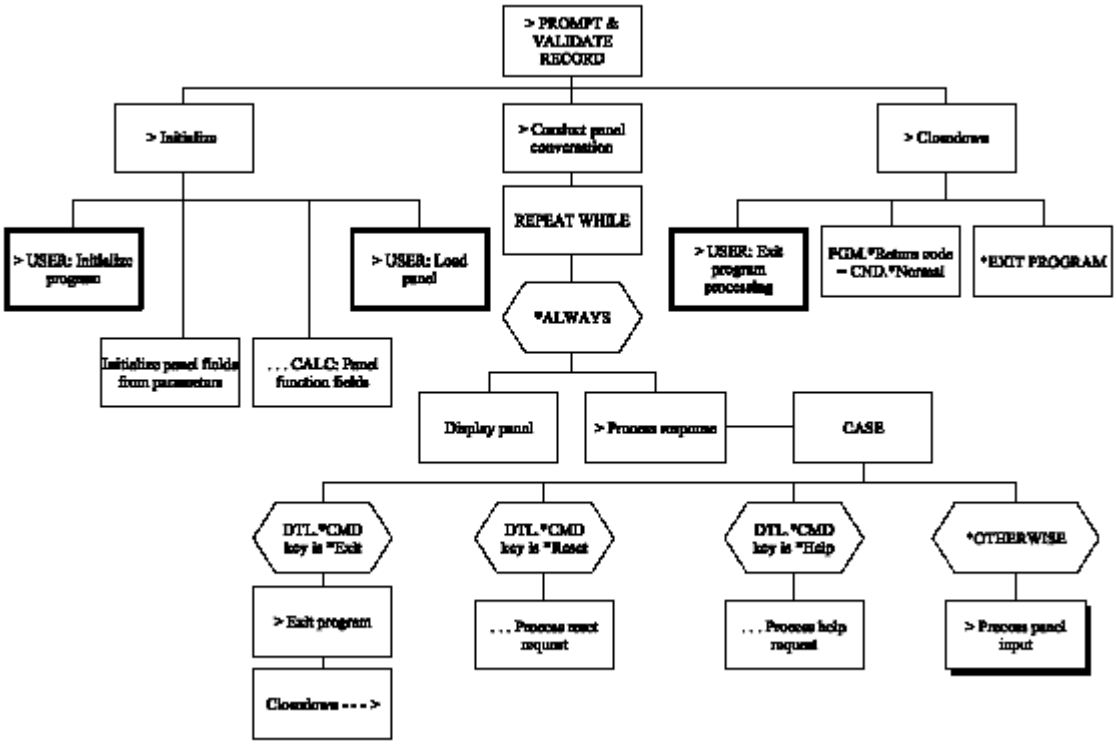
Edit Transaction (Chart 7 of 8)



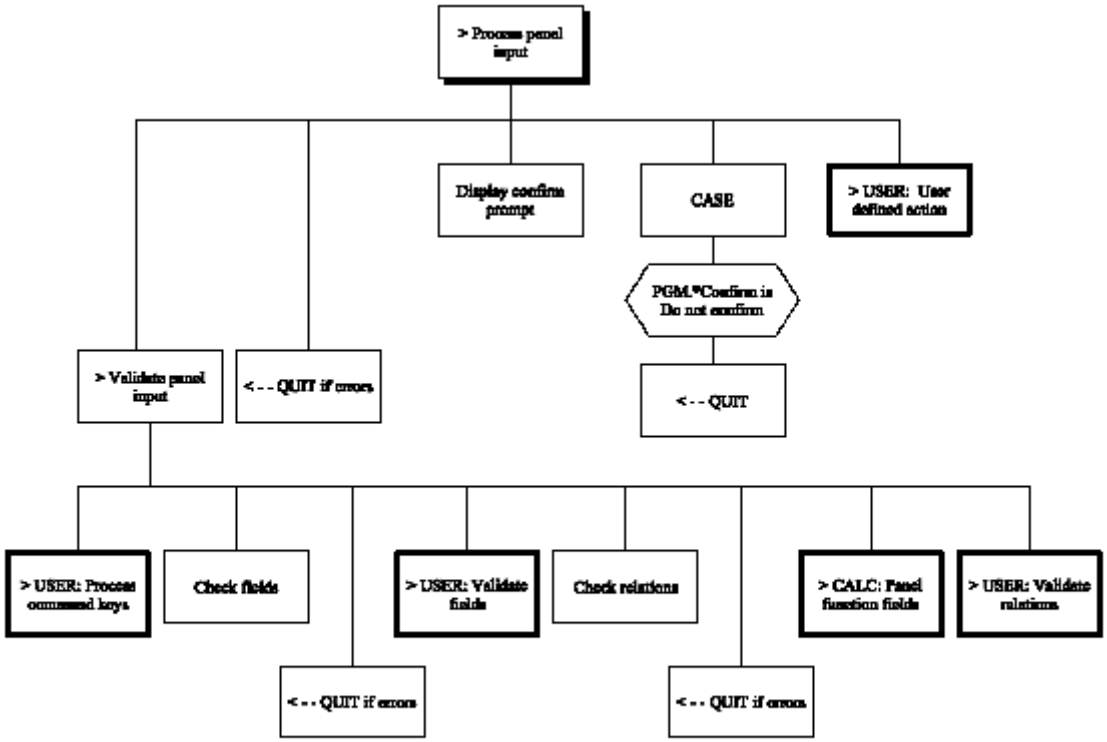
Edit Transaction (Chart 8 of 8)



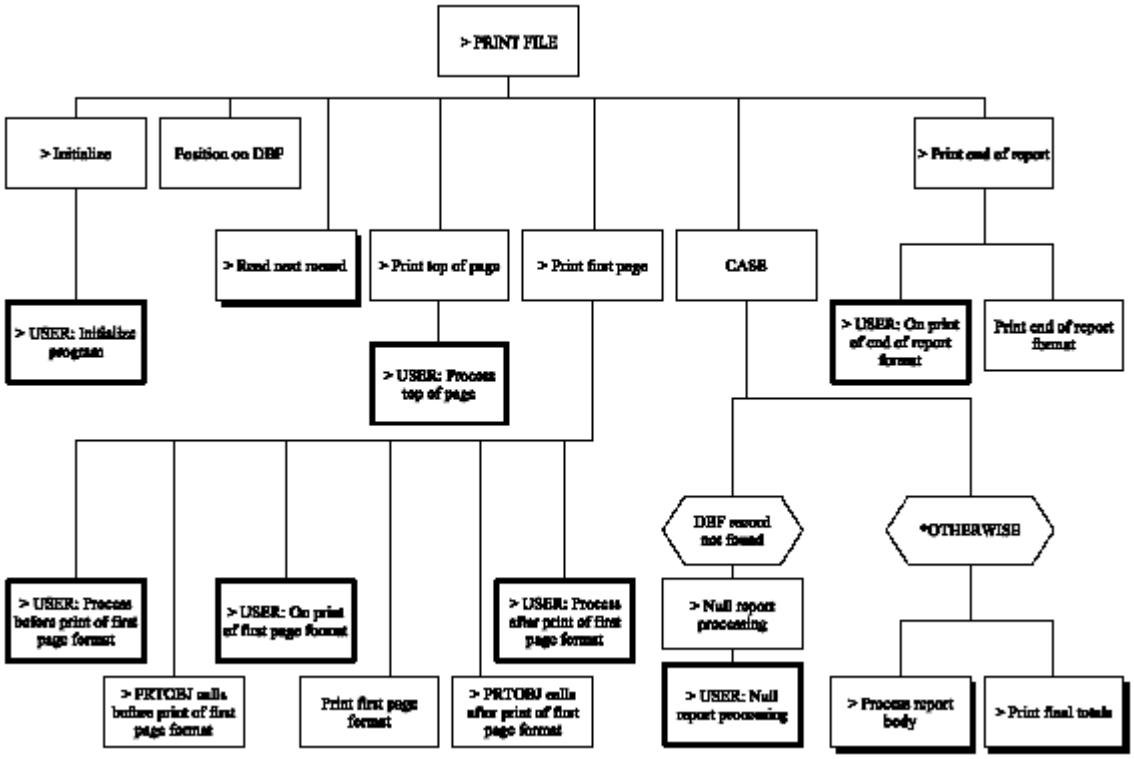
Prompt and Validate Record (Chart 1 of 2)



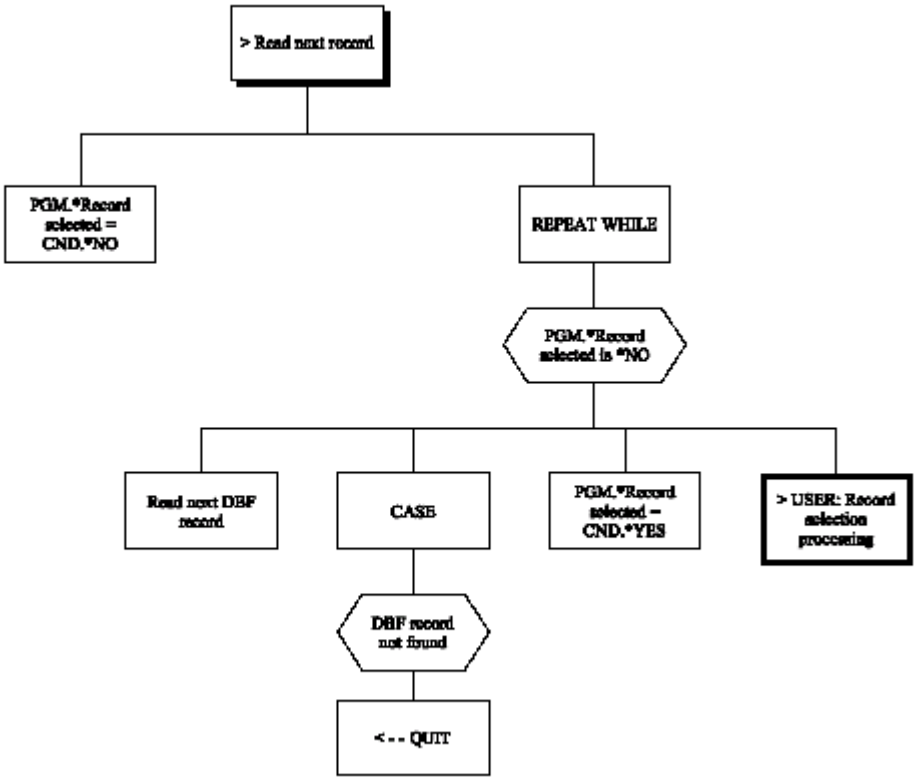
Prompt and Validate Record (Chart 2 of 2)



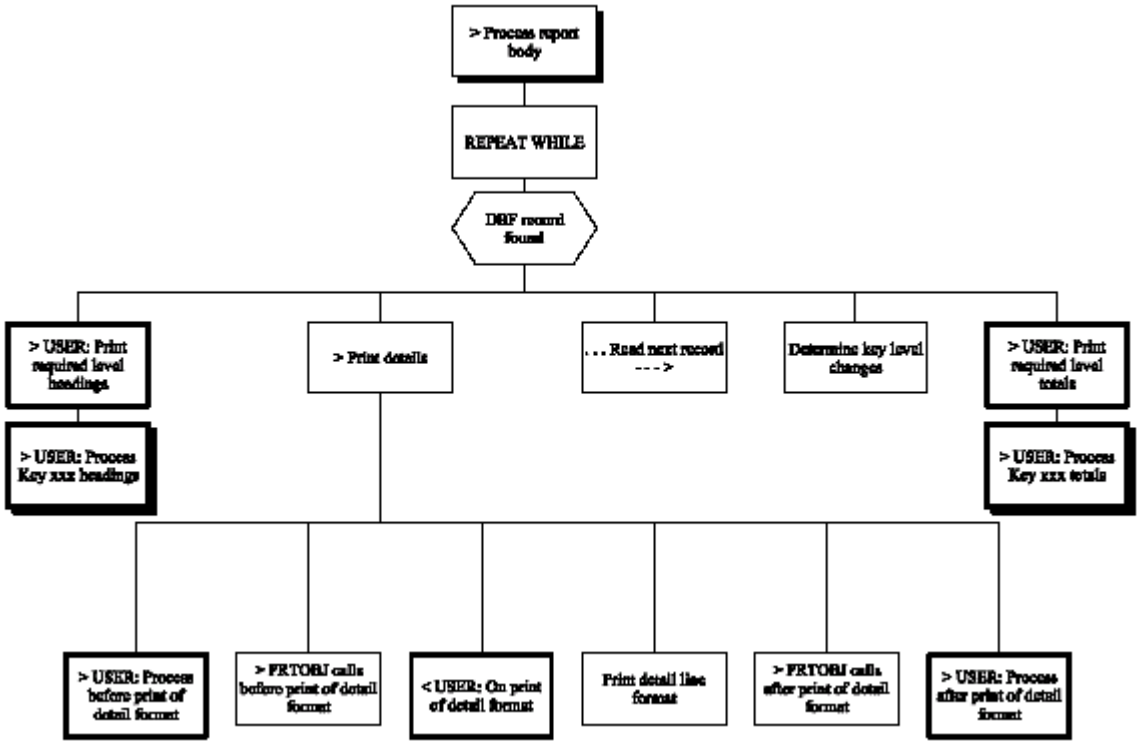
Print File (Chart 1 of 5)



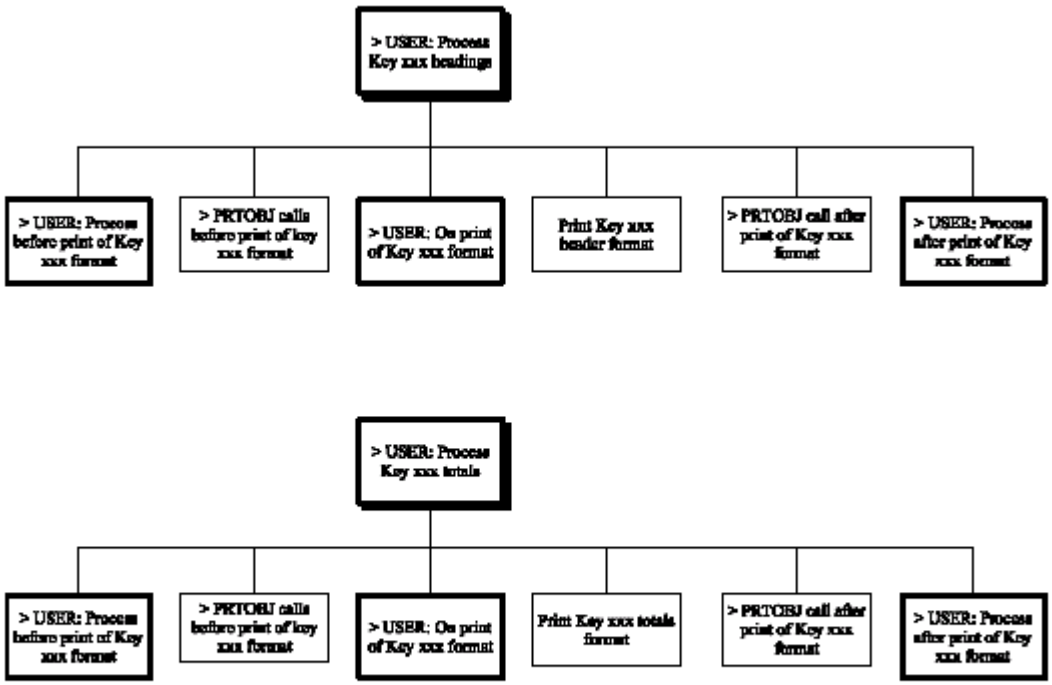
Print File (Chart 2 of 5)



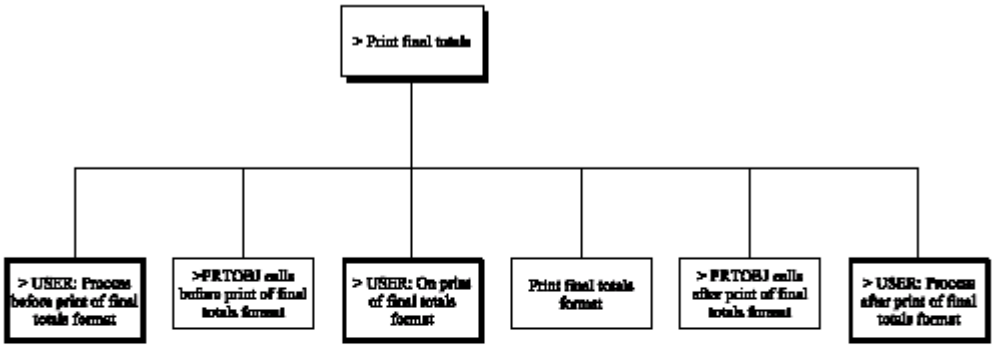
Print File (Chart 3 of 5)



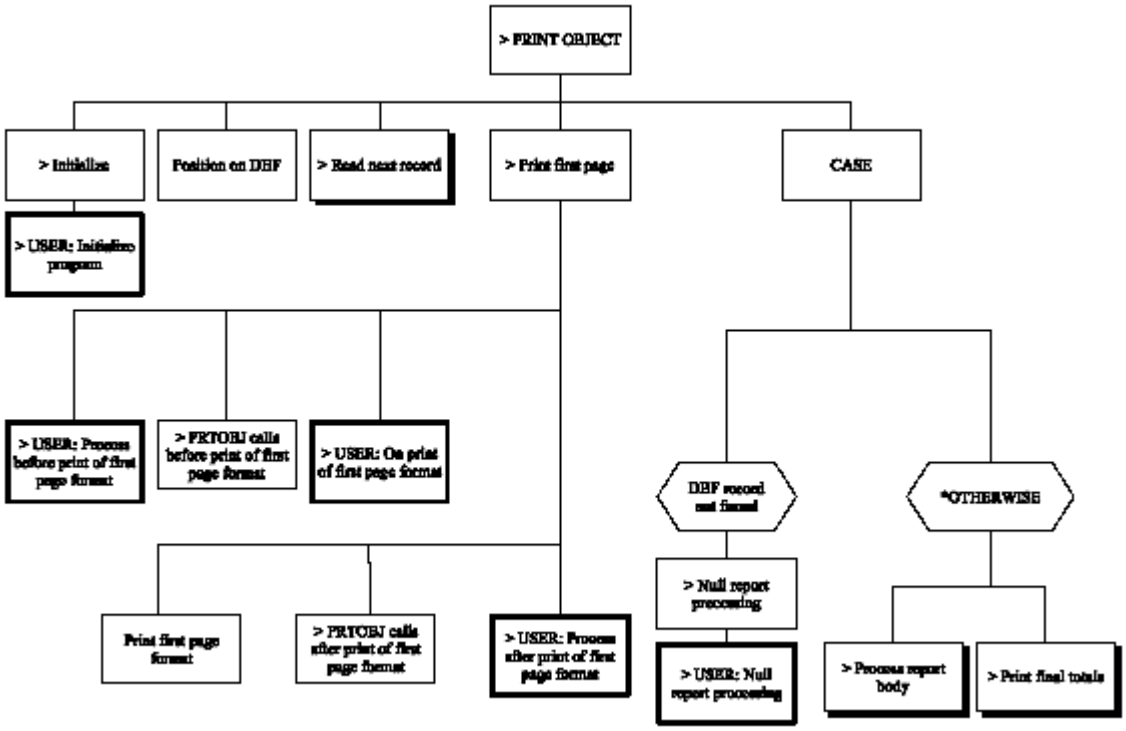
Print File (Chart 4 of 5)



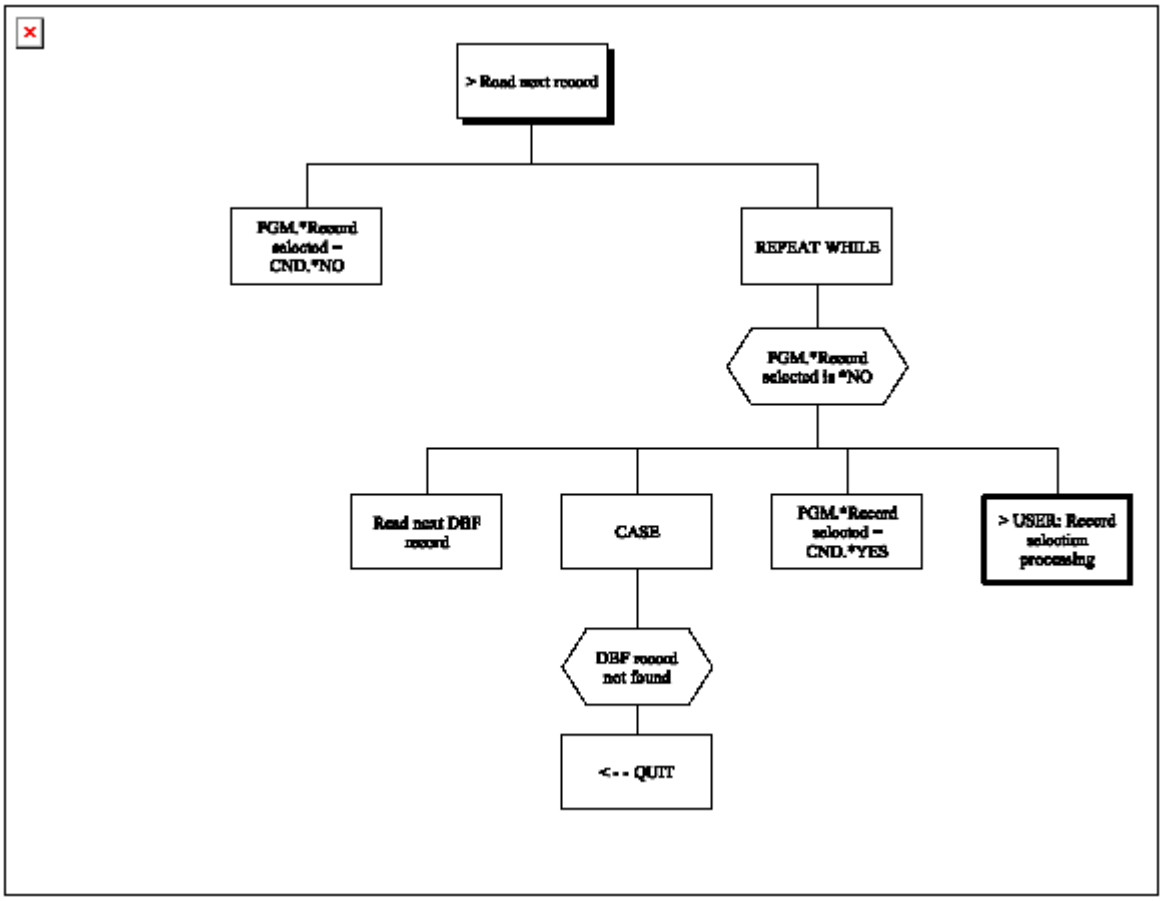
Print File (Chart 5 of 5)



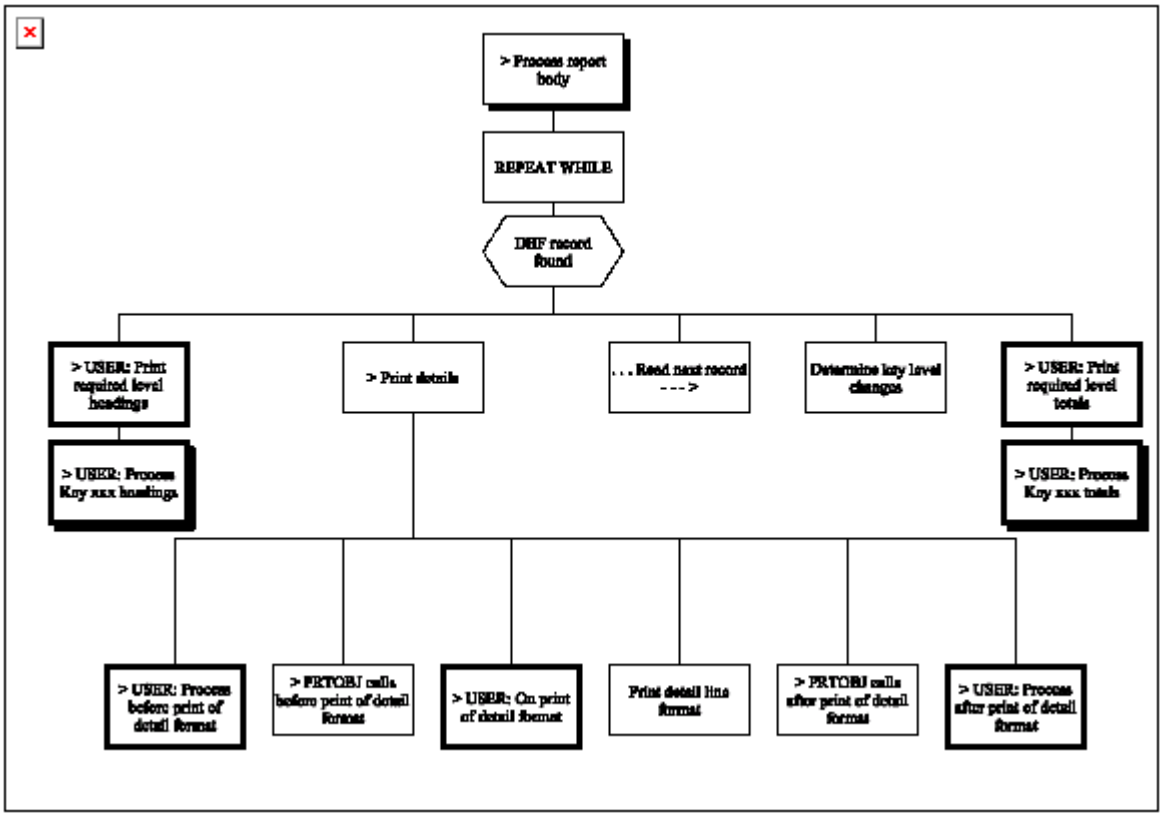
Print Object (Chart 1 of 5)



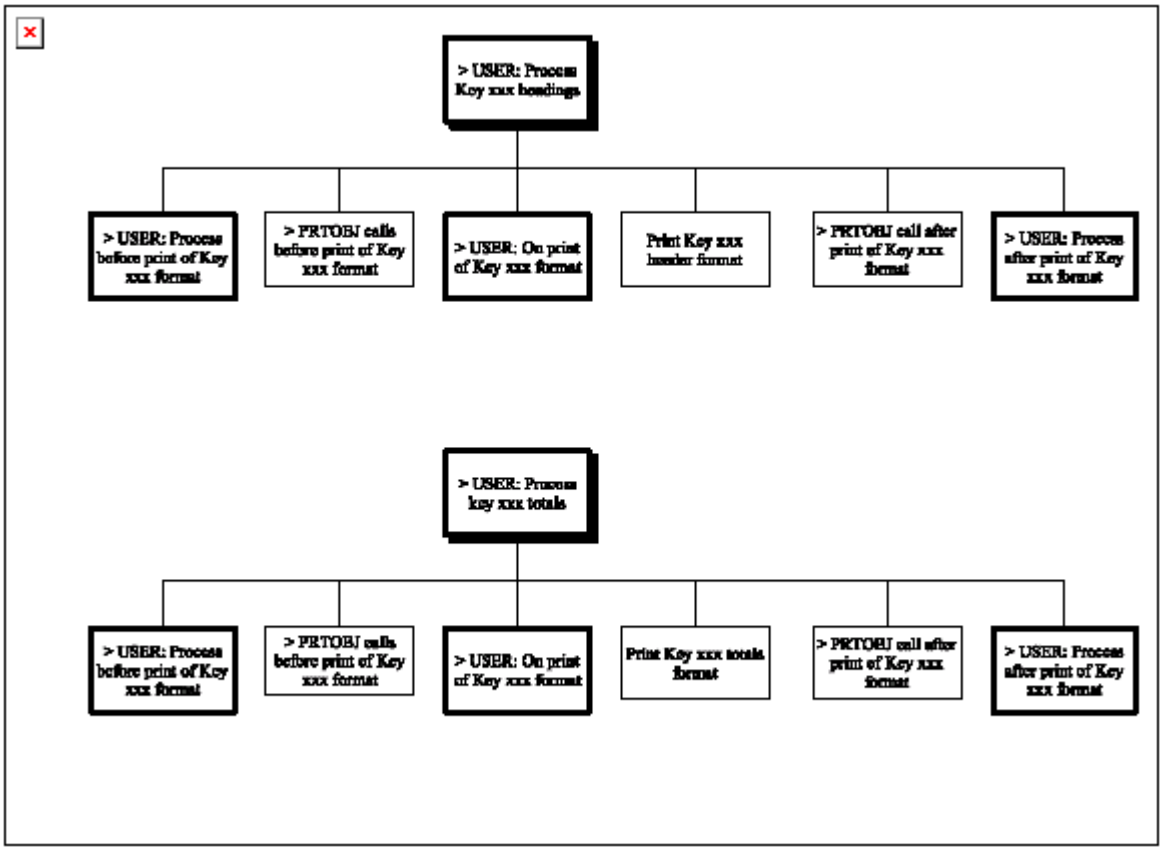
Print Object (Chart 2 of 5)



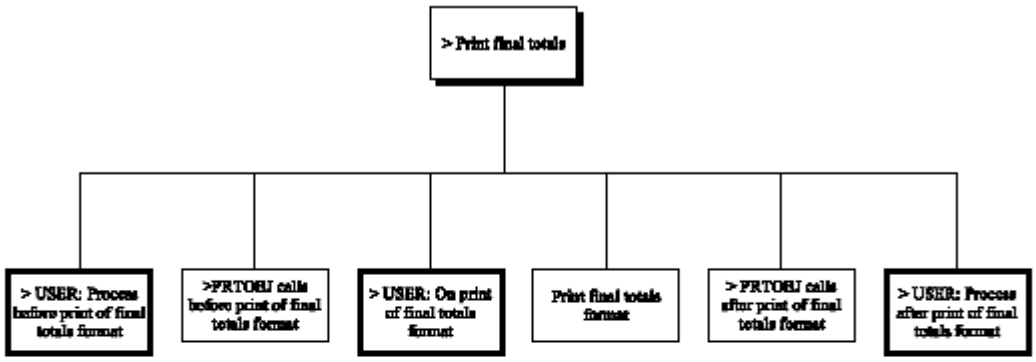
Print Object (Chart 3 of 5)



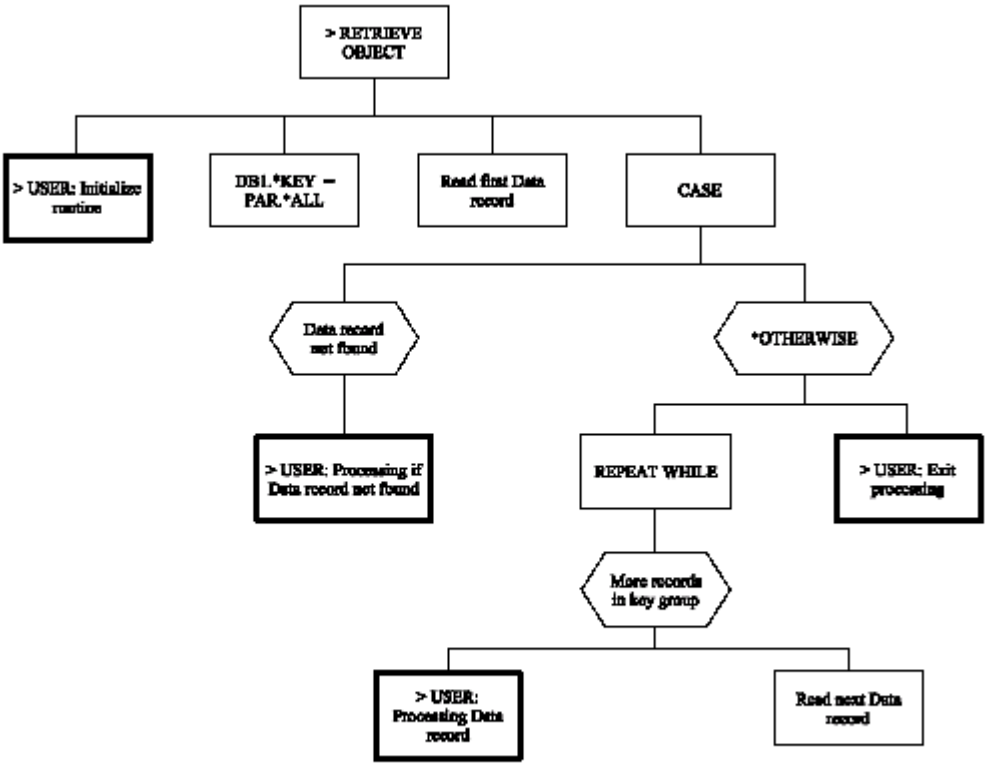
Print Object (Chart 4 of 5)



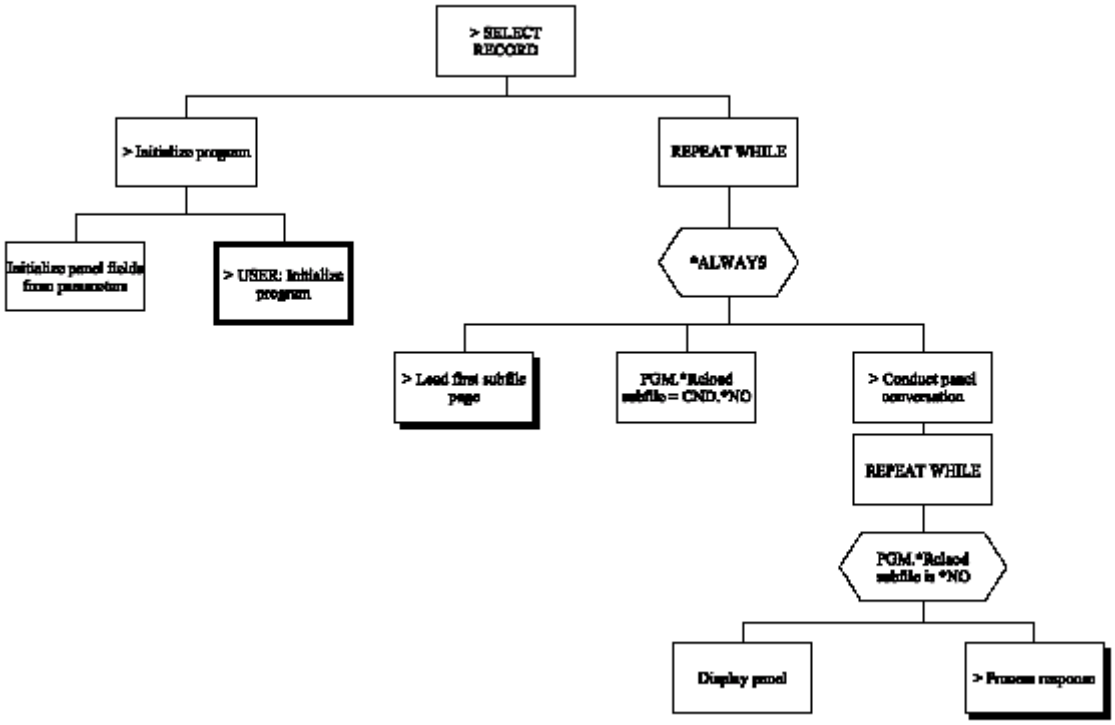
Print Object (Chart 5 of 5)



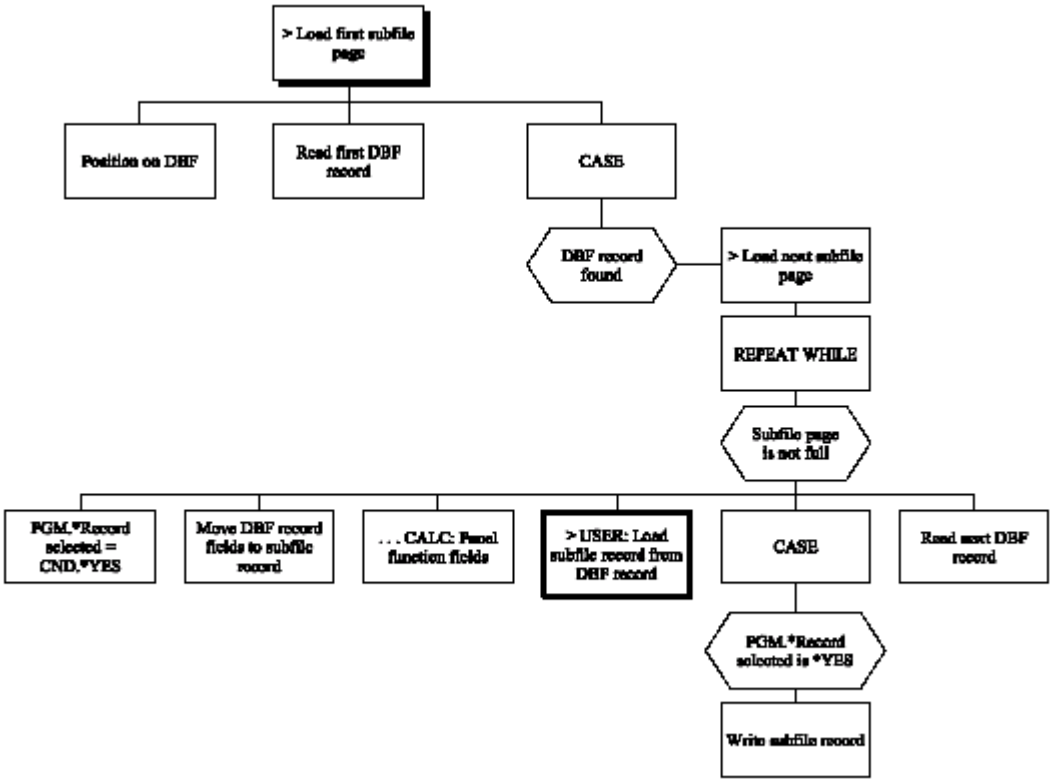
Retrieve Object



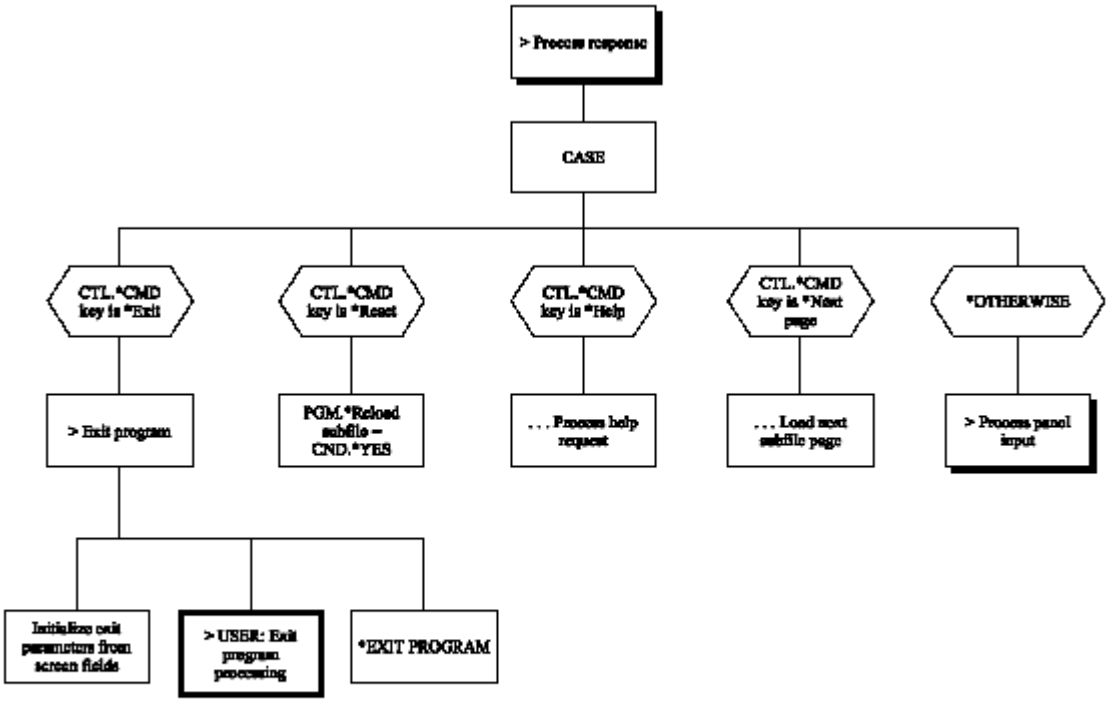
Select Record (Chart 1 of 4)



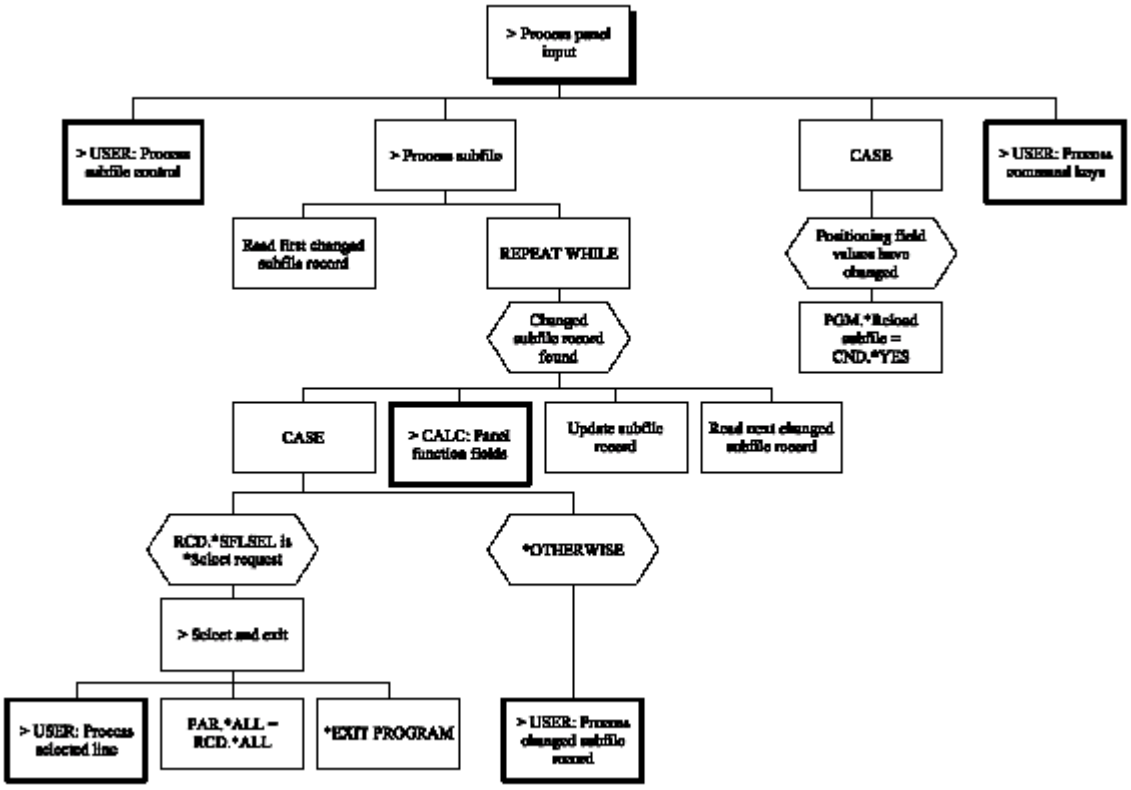
Select Record (Chart 2 of 4)



Select Record (Chart 3 of 4)



Select Record (Chart 4 of 4)



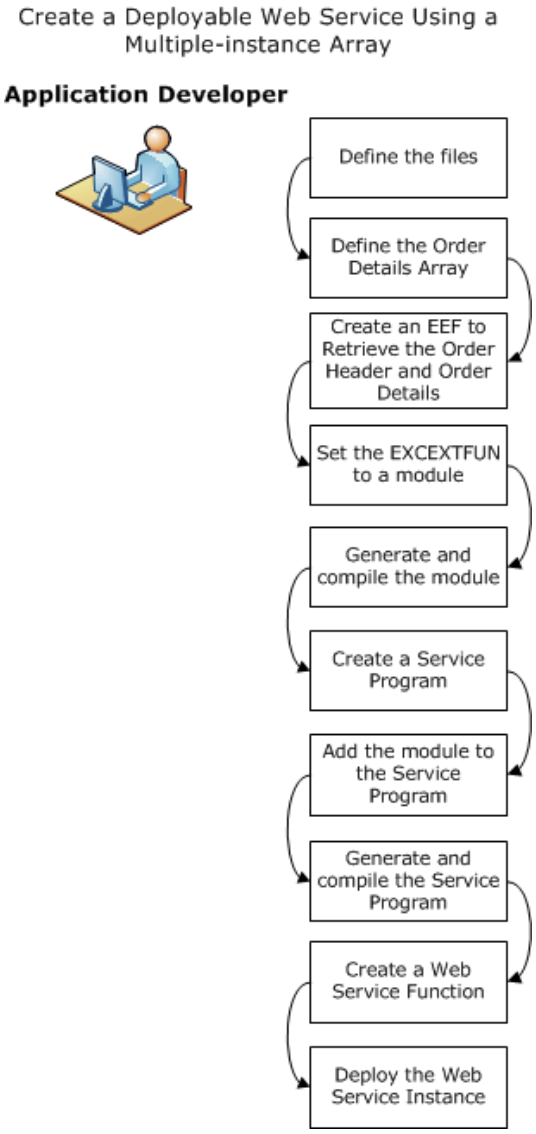
Appendix B: How to Create a Deployable Web Service Using a Multiple-instance Array

This process explains how an experienced CA 2E Application Developer can use CA 2E web service support with enhanced array support to make an invocation and retrieve an order. This process includes the order header and multiple order detail lines.

Important! The coding examples are designed to illustrate enhanced array support and are not *production-ready*. For example, the examples do not contain any user-added error handling.

As an experienced CA 2E Application Developer, you want your result to be a deployable 2E Web Service that uses Enhanced Array Support to return multiple-instances of data in a single invocation. In this scenario, we use an Order Header and Order Detail file, and the order has a single order header and multiple detail lines.

The following illustration shows how you complete the process:



Follow these steps:

1. Define the Files (see page 745)
2. Define the Order Details Array (see page 747)
3. Create an Execute External Function (see page 748)
4. Retrieve the Order Header and Order Details (see page 748)
5. Set the EXCEXTFUN function to a module (see page 762)
6. Generate and compile the module (see page 763)
7. Create a Service Program (see page 764)
8. Add the module to the Service Program (see page 765)
9. Generate and compile the Service Program (see page 766)
10. Create a web service function (see page 767)
11. Deploy the Web Service instance (see page 768)

Note: This procedure uses individual *MOVE ARRAY statements. As you become familiar with the process, the *MOVE ARRAY, *ALL (see page 769) function helps make your work more efficient.

Back to Top

Define the Files

We assume the following basic file definitions for the process:

Note: The field types are not critical to this scenario, so you can define field type as you see fit.

Order Header Files

If your screen shows the Order Header File as:

```
Order header    Known by    Order Number
Order header    Has         Customer Name
Order header    Has         Order Date
```

Then the Order Header Records look like this example:

Order Number	Customer Name	Order Date
10001	John Doe	123111

Order Number	Customer Name	Order Date
10002	Jane Doe	022811

Order Detail File

If your screen shows the Order Detail File as:

```
Order detail        Owned by        FIL Order Header
Order detail        Known By        FLD Line Number
Order detail        Refers to               FIL Product
Order detail        Has             FLD Quantity
```

Then the Order Detail Records look like this example:

Order Number	Line Number	Product Code	Quantity
10001	1	BH	101
10001	3	IP5	103
10002	1	BH	201
10002	3	LH	203

Note: Product code refers to the product file. For example, LH refers to Little Hammer and BH refers to Big Hammer.

Product File

```
FIL Product         Known by        FLD Product code
FIL Product         Has             FLD Product description
FIL Product         Has             FLD Product price date
FIL Product         Has             FLD Product price
```

Product code	Product description	Product Date	Product Price
BH	Big Hammer	20010101	23.45
LH	Little Hammer	20101010	18.97
IP5	Ice Pick	20020202	299.00

Back to Top

Define the Order Details Array

We assume the following array definition for the process.

Note: The Element number field should be of type NBR.

Follow these steps:

1. Create an array named Order Details over the *Arrays file.
2. Take option Z to open the EDIT ARRAY DETAILS screen.
3. Define the Order details array with the retrieval index structure of the order detail file and additionally an Element number field (Type NBR), as follows:

```
                                Op: COCSI01     QPADEV0007   1/25/12 15:24:49
 EDIT ARRAY DETAILS                 SBC86CMDL
Array  . . . . . : Order Details
Number of elements : _100   (1 - 9999)
Sequence . . . . . : A (A=Ascending, D=Descending)
Unique . . . . . : Y (Y=Unique, N=non-unique)

 ?  File/*FIELD              Access path/Field/Array    Seq
 _  Order detail             Retrieval index            1
 _  *FIELD                   Element number             2
```

4. Press F7 to access the EDIT ARRAY KEY ENTRIES panel and define Element number as the key to the array.

```
                                Op: COCSI01     QPADEV0007   1/25/12 15:27:07
 EDIT ARRAY KEY ENTRIES             SBC86CMDL
Array  . . . . . : Order Details          Sequence . . . . . : ASCEND
Unique . . . . . : UNIQUE                 Number of elements :   100

                                GEN                          Key   Par Seq
 ? Field                        Name       Type Length       no.   Grp No.
 _ Product code              CDE AECD       A       16             1
 _ Order number              NBR ABDZ       A        5             1
 _ line number               NBR AKNB       A        5             1
 _ quantity                  NBR ALNB       A        5             1
 _ Element number            NBR AMNB       K        5       1     2
```

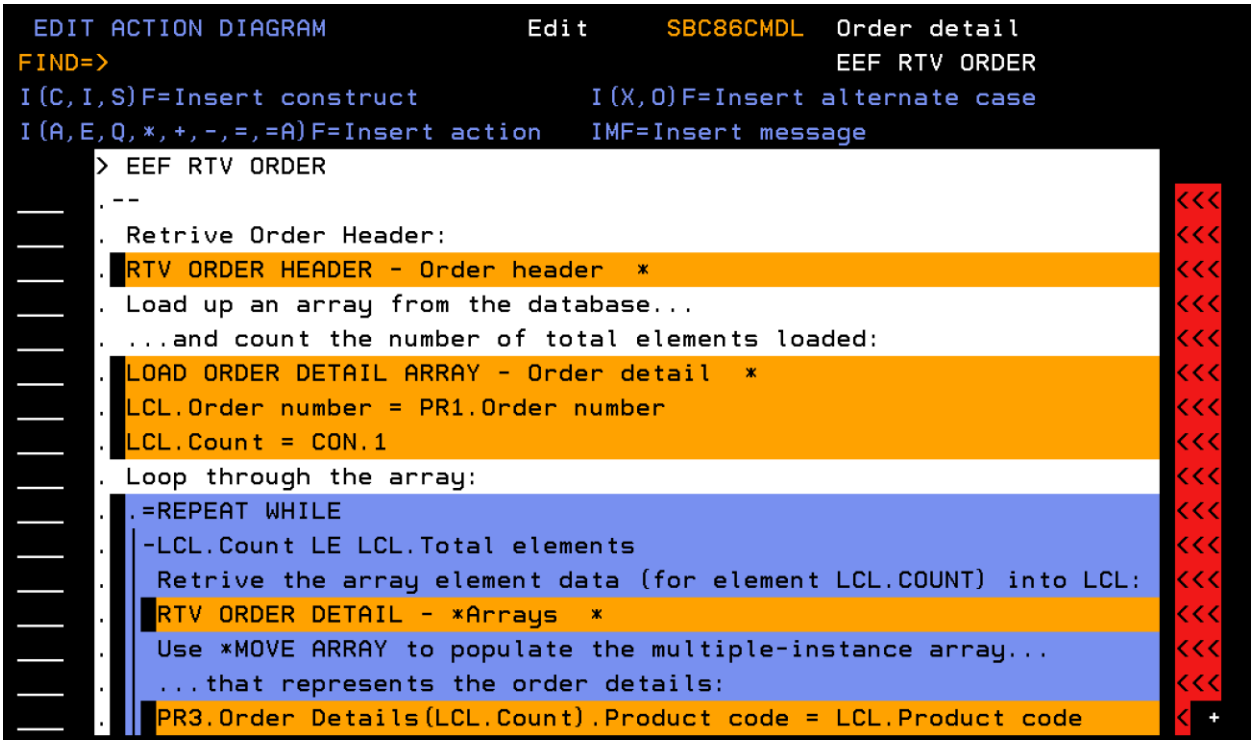
Back to Top

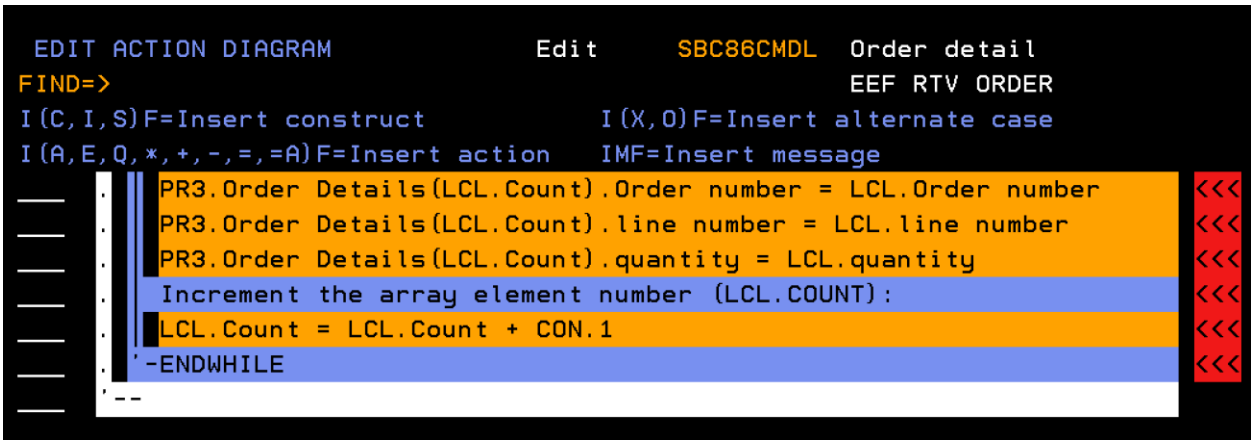
Create an EXCEXTFUN to Retrieve the Order Header and Order Details

The function EEF RTV ORDER, EXCEXTFUN type, based over the Order Detail file takes an input parameter of an order number and returns order information as output parameters as follows:

- The order header is represented by a single-instance output parameter.
- The order detail lines are represented by a multiple-instance output parameter.

This section describes how to define the EXCEXTFUN to include parameter interface and action diagram business logic, including calls to other functions. After you complete the steps in this section, your Action Diagram should match these example screens:





Important! As a CA 2E developer, you need to understand the architectural distinction between the two mechanisms to manipulate array data, despite the ability to use a common structural definition:

- Data can exist and be modified in an array by using database functions (Create Object – CRTOBJ, Delete Object –DLTOBJ, Change Object – CHGOBJ, and Retrieve Object – RTVOBJ) based over the *Arrays file. However, this array data cannot be accessed by the *MOVE ARRAY function.
- Back to TopData can also exist and be modified in a multiple-instance array parameter (in the PAR context) and in the ARR context by using the *MOVE ARRAY function. However, that array data cannot be accessed by database functions (Create Object – CRTOBJ, Delete Object –DLTOBJ, Change Object – CHGOBJ, and Retrieve Object – RTVOBJ) based over the *Arrays file.

Note: The function EEF RTV ORDER (Order Detail) relies on four other internal functions, as listed in the following table.

Function Name	Function Type	Based over file
RTV ORDER HEADER	RTVOBJ	Order header
RTV ORDER DETAIL	RTVOBJ	*Arrays
CRT ORDER DETAIL	CRTOBJ	*Arrays
LOAD ORDER DETAIL ARRAY	RTVOBJ	Order detail
EEF RTV Order	EXCEXTFUN	Order detail

You need to build the required internal functions and the external function EEF RTV ORDER (Order detail):

Note: The file the functions is based over is shown in parentheses.

Follow these steps:

1. RTV Order Header (Order Header) (see page 750)
2. RTV Order Detail (*Arrays) (see page 752)
3. CRT Order Detail (*Arrays) (see page 753)
4. Load Order Detail Array (Order detail) (see page 754)
5. EEF RTV Order (Order detail) (see page 757)

Back to Top

Retrieve the Order Header

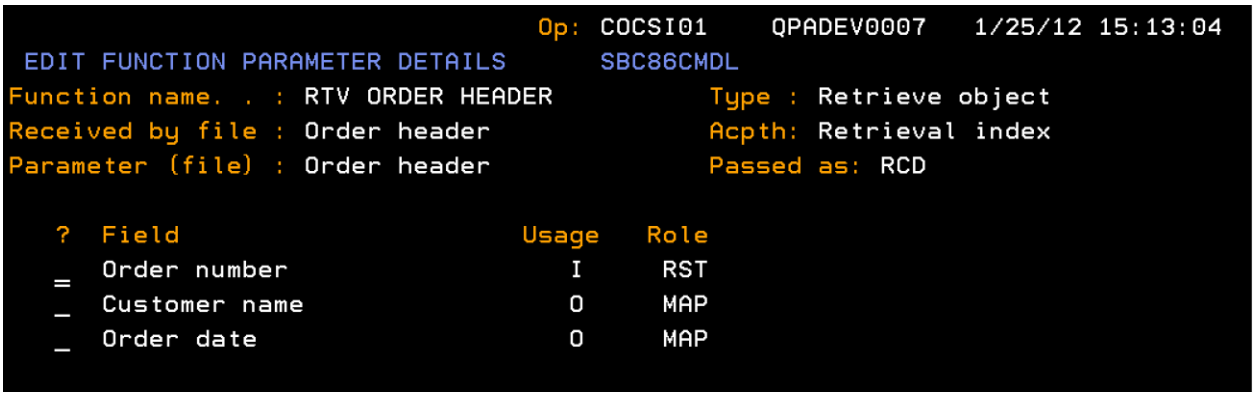
This section explains how you retrieve information about an order header record, given an order number.

Follow these steps:

1. Define the RTV ORDER HEADER: Parameter interface as follows:

```
                                Op: COCSI01    QPADEV0007   1/25/12 15:12:49
 EDIT FUNCTION PARAMETERS            SBC86CMDL
Function name. . : RTV ORDER HEADER          Type : Retrieve object
Received by file : Order header              Acpth: Retrieval index
                                                    Passed      Pgm  Par
 ?  File/*FIELD              Access path/Field/Array        Seq  Ctx  Ctx A
 =  Order header             Retrieval index          RCD  ___
 _  _________________________ _________________________  ___  ___
```

2. Define the function parameter details as follows:



3. Configure an action diagram: RTV ORDER HEADER as follows:

   This example shows User Points that are not empty.

4. Save and exit the function.

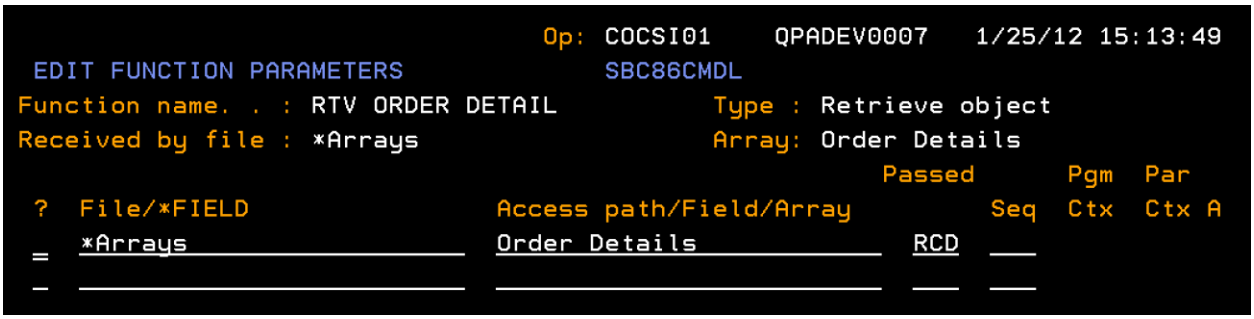
Back to Top

RTV Order Detail (*Arrays)

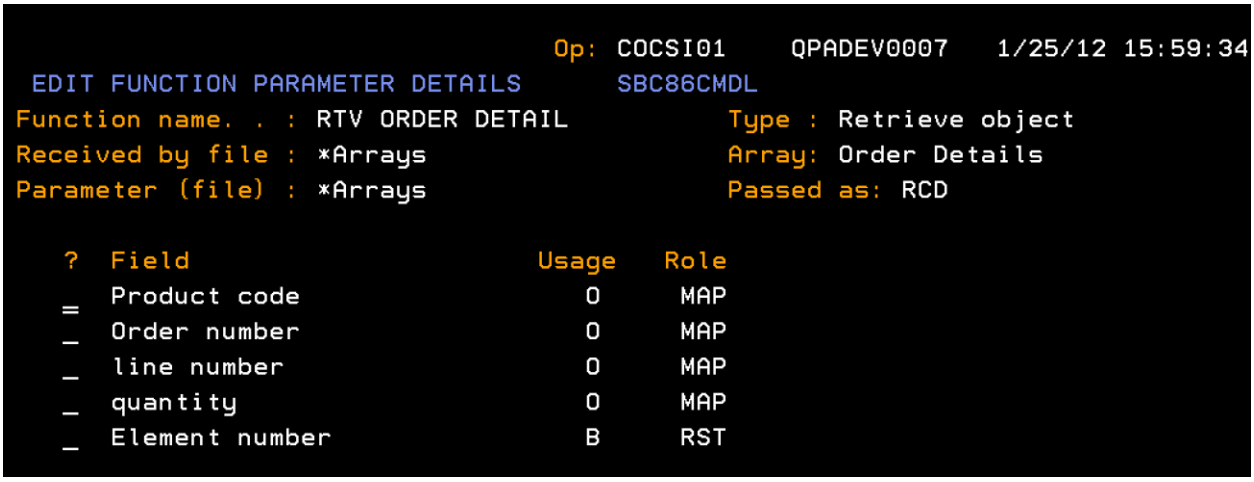
This section describes how you retrieve the data for a single element in the Order Details array. You need to configure the RTV ORDER DETAIL to retrieve the data for a single element in the Order Details array.

Follow these steps:

1. Configure the RTV ORDER DETAIL: Parameter interface as follows:



2. Define the parameter details as follows:



3. Configure the RTV ORDER DETAIL: Action Diagram as follows:

4. Save and exit the function.

Back to Top

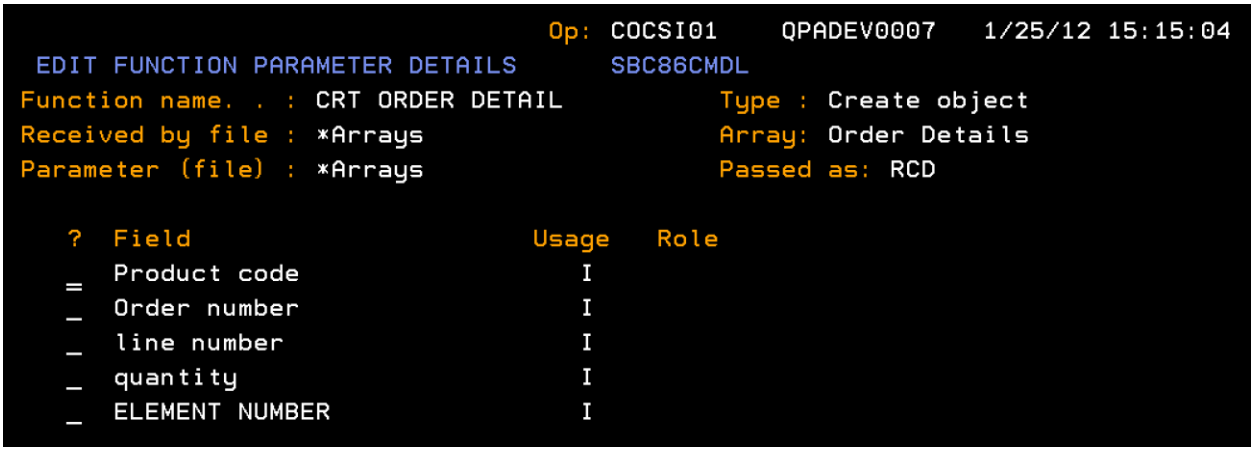
CRT Order Detail (*Arrays)

This section describes how you configure the CRT ORDER DETAIL to populate the data for a single element in the Order Details array.

Follow these steps:

1. Configure the CRT ORDER DETAIL: Parameter interface as follows:

2. Define the parameter details as follows:



Note: The CRT ORDER DETAIL contains no user-added processing in the action diagram.

3. Save and exit the function.

Back to Top

Load Order Detail Array (Order detail)

This section explains how to define a function to populate the Order Detail array. The processing runs through the database to find all order detail lines for a given order number. Each detail line is added as a single element in the Order Details array.

Follow these steps:

1. Create a function called LOAD ORDER DETAIL ARRAY, and configure the parameter interface as follows:

   Note: Define the field Total elements as type NBR.

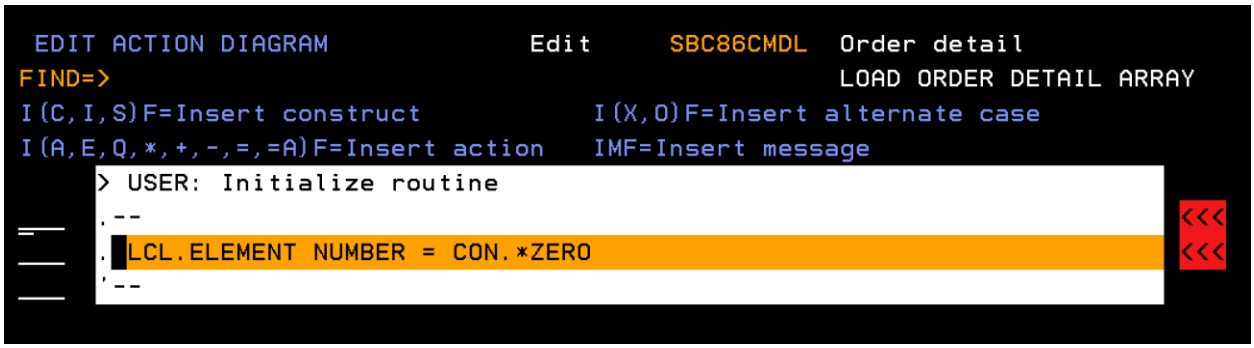
```
                                Op: COCSI01    QPADEV0007   1/25/12 15:15:21
 EDIT FUNCTION PARAMETERS            SBC86CMDL
Function name. . : LOAD ORDER DETAIL ARRAY   Type : Retrieve object
Received by file : Order detail              Acpth: Retrieval index
                                                    Passed      Pgm  Par
 ?  File/*FIELD              Access path/Field/Array        Seq  Ctx  Ctx A
 =  Order detail             Retrieval index          KEY  ___
 _  *FIELD                   Total elements           FLD  ___
```

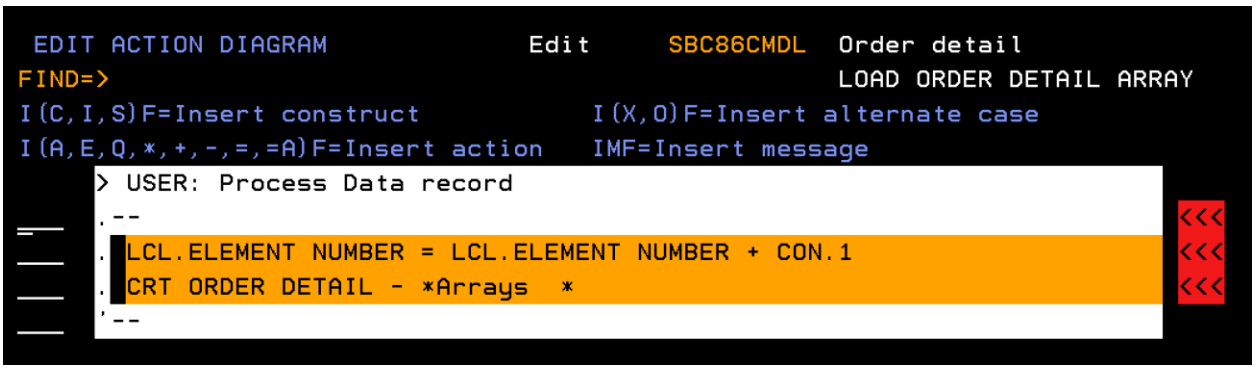
2. Define the parameter details as follows:

```
                                Op: COCSI01    QPADEV0007   1/25/12 15:15:31
 EDIT FUNCTION PARAMETER DETAILS     SBC86CMDL
Function name. . : LOAD ORDER DETAIL ARRAY   Type : Retrieve object
Received by file : Order detail              Acpth: Retrieval index
Parameter (file) : Order detail              Passed as: KEY

   ?  Field                    Usage   Role
   =  Order number               I      RST
   _  line number                I      POS
```

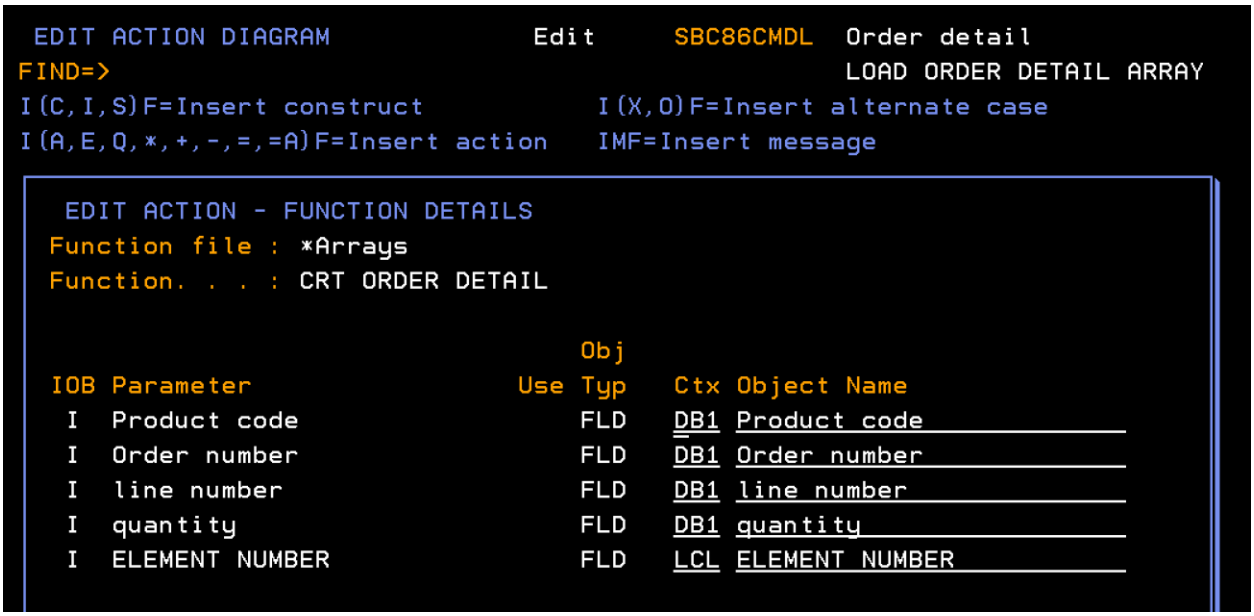
```
                                Op: COCSI01    QPADEV0007   1/25/12 15:15:40
 EDIT FUNCTION PARAMETER DETAILS     SBC86CMDL
Function name. . : LOAD ORDER DETAIL ARRAY   Type : Retrieve object
Received by file : Order detail              Acpth: Total elements
Parameter (file) : *FIELD                    Passed as: FLD

   ?  Field                    Usage   Role
   =  Total elements             O      MAP
```

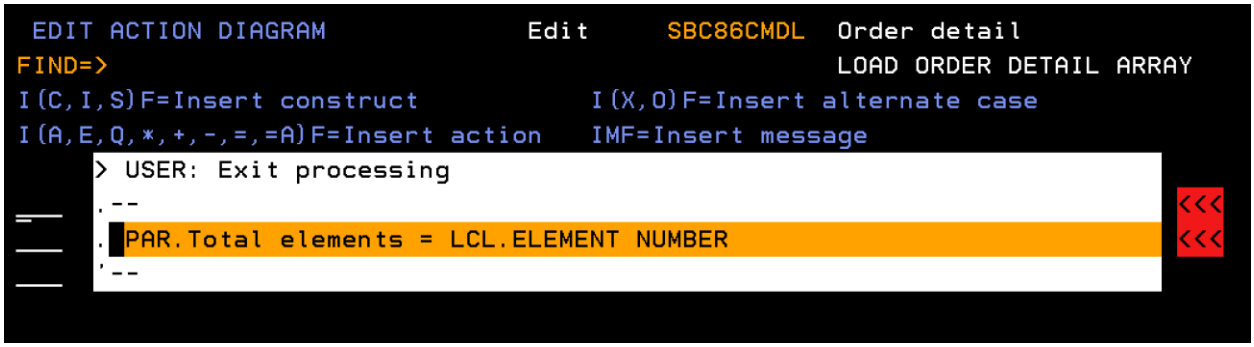
3. Configure a LOAD ORDER DETAIL ARRAY: Action Diagram as follows:





4. Call to the CRT ORDER DETAIL as follows:





5. Save and exit the function.

Back to Top

EEF RTV Order (Order detail)

This section describes how you define the external function EEF RTV Order (Order detail) including the parameter interface and action diagram business logic. The EEF RTV Order (Order detail) retrieves information about an order. Given an input of an order number, the function retrieves information about the order header, and all of the order detail lines. The order detail lines are represented using a multiple-instance array parameter.

The key concept is to populate the multiple-instance array parameter (PR3) with order detail lines from the database. Typically, you can obtain database information using a function of type RTVOBJ. However, the RTVOBJ cannot directly access the (PR3) parameters of the EEF. A multiple-instance array parameter can only be defined on functions of Execute External Function and Execute User Program.

A RTVOBJ call might be placed in a loop to retrieve one order detail line at a time into the, LCL context for example, and then the information could be moved from the LCL context to an element of the multiple-instance array (PR3) on the EEF. However, multiple calls to a RTVOBJ based over a database access path is undesirable from a performance point of view. Moreover, if the order detail lines do not exist with some contiguous key or positioned, additional code will have to ensure that each order detail line is retrieved.

We address this issue as follows:

From the AD of the EEF call a RTVOBJ, named LOAD ORDER DETAIL ARRAY, based over a database access path (Order detail) one time, and for each order detail record, create an element in an array.

Note: The array has the same structure as the order detail record, but has an additional key field of Element number.

Once the Array is created, a loop in the EEF can step through the array and call a RTVOBJ, named RTV ORDER DETAIL, based over the Order Detailsarray to recall one element at a time to the LCL context.

The data in the LCL context that represents a single order detail record can then be populated into multiple-instance array parameter (PR3) of the EEF that represents the order detail lines. This move from the single-instance LCL context to a multiple-instance context can be achieved with the *MOVE ARRAY function.

This approach has the benefit that the processing is faster than multiple RTVOBJ (database) calls and the looping through the array in memory overcomes any gaps between order detail line numbers.

The EXCEXTFUN function type can be deployed as a *MODULE, which allows it to be bundled into Service Program and deployed as a web service. Thus a single invocation of the web service can retrieve all the information regarding the order header and all order detail lines.

Follow these steps:

1. Create a function called EEF RTV Order and define the parameter interface as follows:

   Notes:

   - EEF RTV Order has function option Duplicate parameters set to Yes.
   - PR3 is defined as a multiple-instance array parameter.

   PR1

   Indicates the input parameter for the Order number, passed as a field (FLD), from which the Header and Detail are retrieved.

   PR2

   Indicates the output parameter for the Order Header, passed as a record format (RCD), which is populated with the single-instance data for the Order Header.

   PR3

   Indicates the output parameter for the Order details array.

   Note: This parameter is marked as being a multiple-instance parameter, so multiple order detail records can be contained within this one parameter. If the A(Pass as Array) field is marked as Y, before usages have been specified on the EDIT FUNCTION PARAMETER DETAILS panel for the fields of this parameter a warning message will be sent: Y2V0717 A (Pass as Array) cannot be set to 'Y'.However, you may leave the field set to Y and zoom into the parameter to specify usages.

```
                                Op: COCSI01    QPADEV0007   1/25/12 15:07:59
 EDIT FUNCTION PARAMETERS           SBC86CMDL
Function name. . : EEF RTV ORDER              Type : Execute external function
Received by file : Order detail               Acpth: *NONE
                                                    Passed       Pgm  Par
 ?  File/*FIELD              Access path/Field/Array          Seq  Ctx  Ctx A
 =  *FIELD                   Order number              FLD    1  PR1  PR1  _
 _  Order header             Retrieval index           RCD    2  PR2  PR2  _
 _  *Arrays                  Order Details             RCD    3  PR3  PR3  Y
 _  _________________________ _________________________ ___  ___             _
```

2. Define the parameter details as follows:

```
                                  Op:  COCSI01    QPADEV0007   1/25/12 15:08:35
 EDIT FUNCTION PARAMETER DETAILS      SBC86CMDL
Function name. . : EEF RTV ORDER              Type : Execute external function
Received by file : Order detail               Acpth: Order number
Parameter (file) : *FIELD                     Passed as: FLD

   ?  Field                       Usage   Role   Flag error
   =  Order number                  I
```

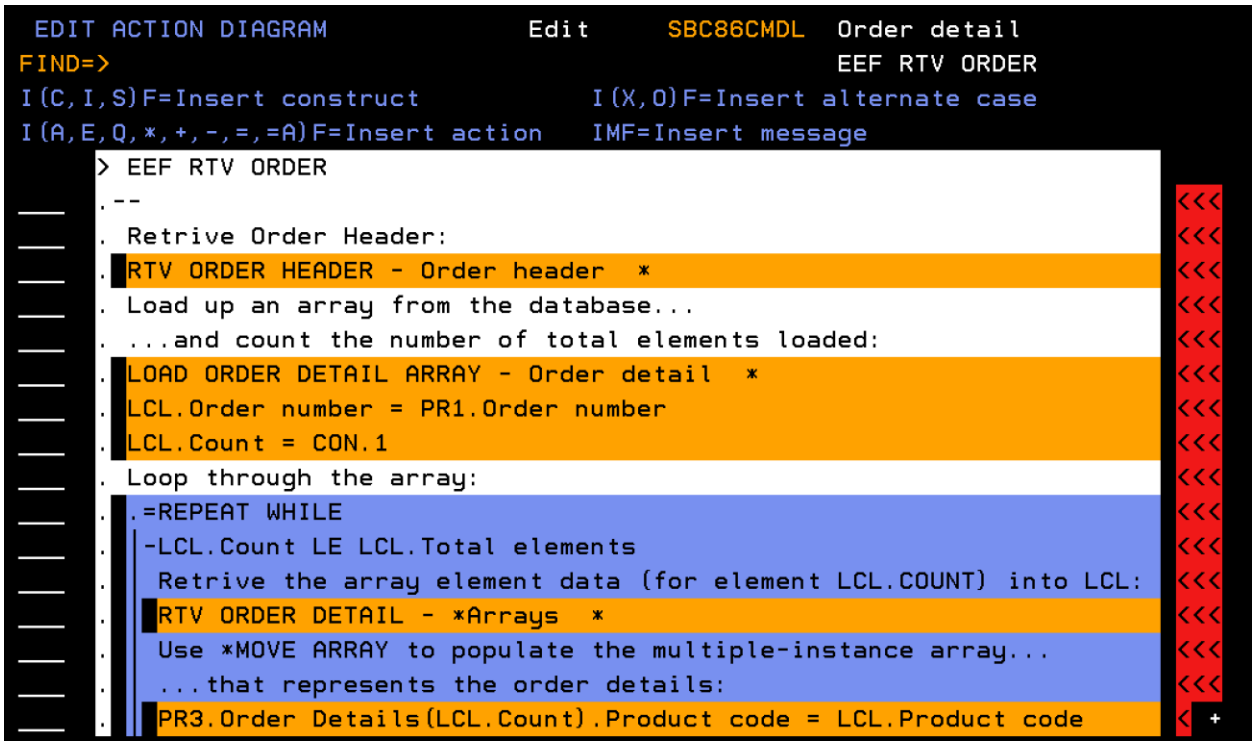
```
                                  Op:  COCSI01    QPADEV0007   1/25/12 15:08:45
 EDIT FUNCTION PARAMETER DETAILS      SBC86CMDL
Function name. . : EEF RTV ORDER              Type : Execute external function
Received by file : Order detail               Acpth: Retrieval index
Parameter (file) : Order header               Passed as: RCD

   ?  Field                       Usage   Role   Flag error
   =  Order number                  O      MAP
   _  Customer name                 O      MAP
   _  Order date                    O      MAP
```

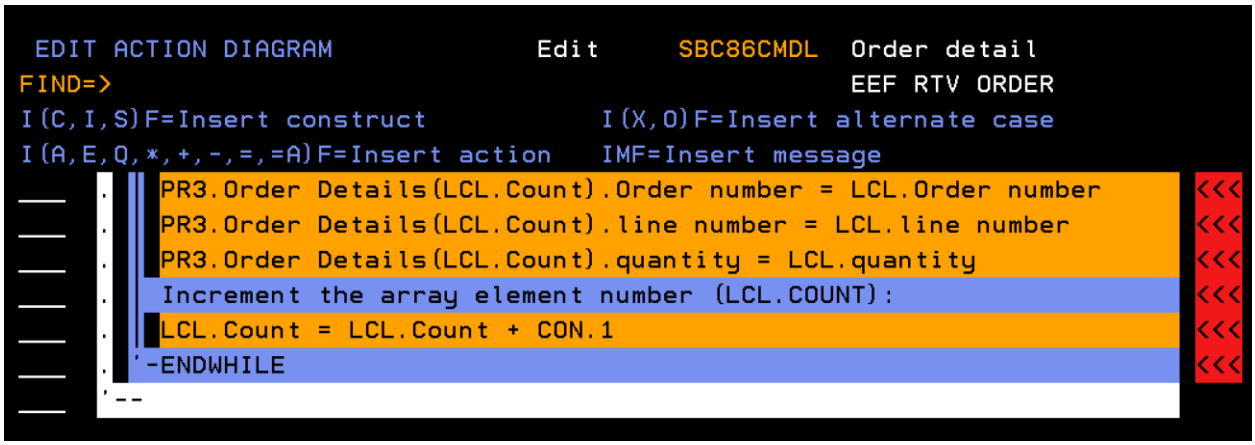
```
                                  Op:  COCSI01    QPADEV0007   1/25/12 15:09:00
 EDIT FUNCTION PARAMETER DETAILS      SBC86CMDL
Function name. . : EEF RTV ORDER              Type : Execute external function
Received by file : Order detail               Array: Order Details
Parameter (file) : *Arrays                    Passed as: RCD (ARRAY)
                                              Number of elements :   100
   ?  Field                       Usage   Role   Flag error
   =  Product code                  O      MAP
   _  Order number                  O      MAP
   _  line number                   O      MAP
   _  quantity                      O      MAP
   _  ELEMENT NUMBER                O      MAP
```

3. Configure the EEF RTV ORDER : Action Diagram as follows:

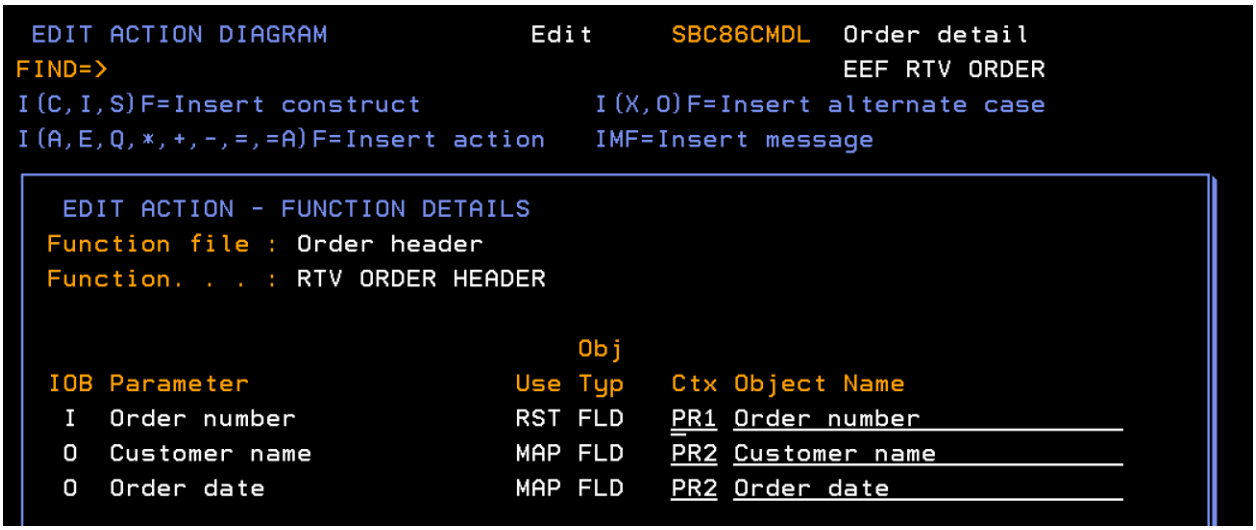
   Notes:

   - Function calls that are not self-evident are shown in the following examples.
   - The Count field should be defined as type NBR.

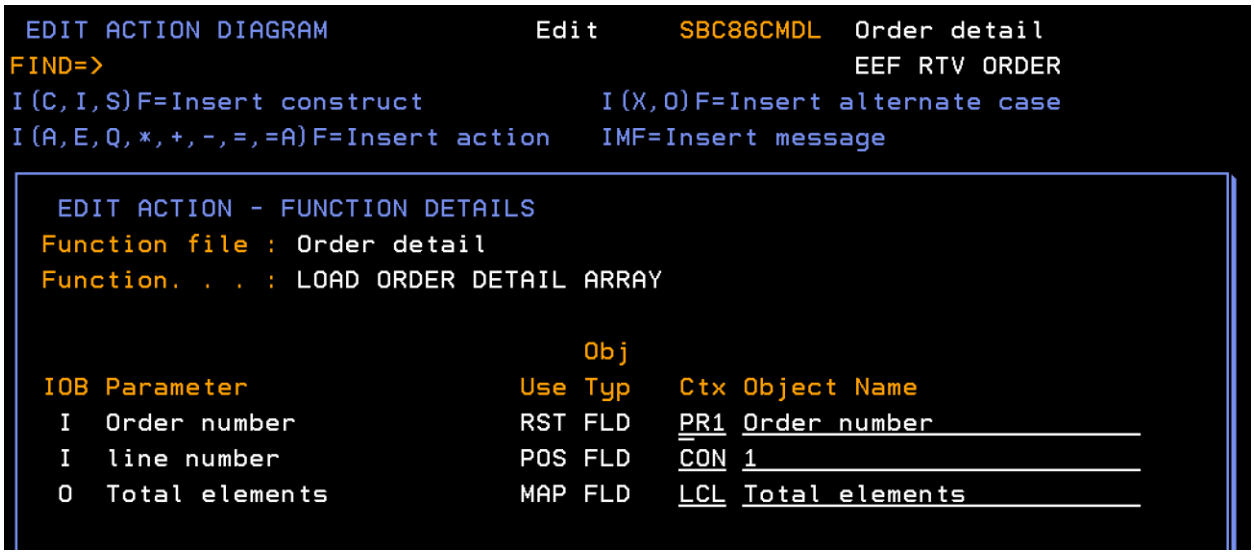




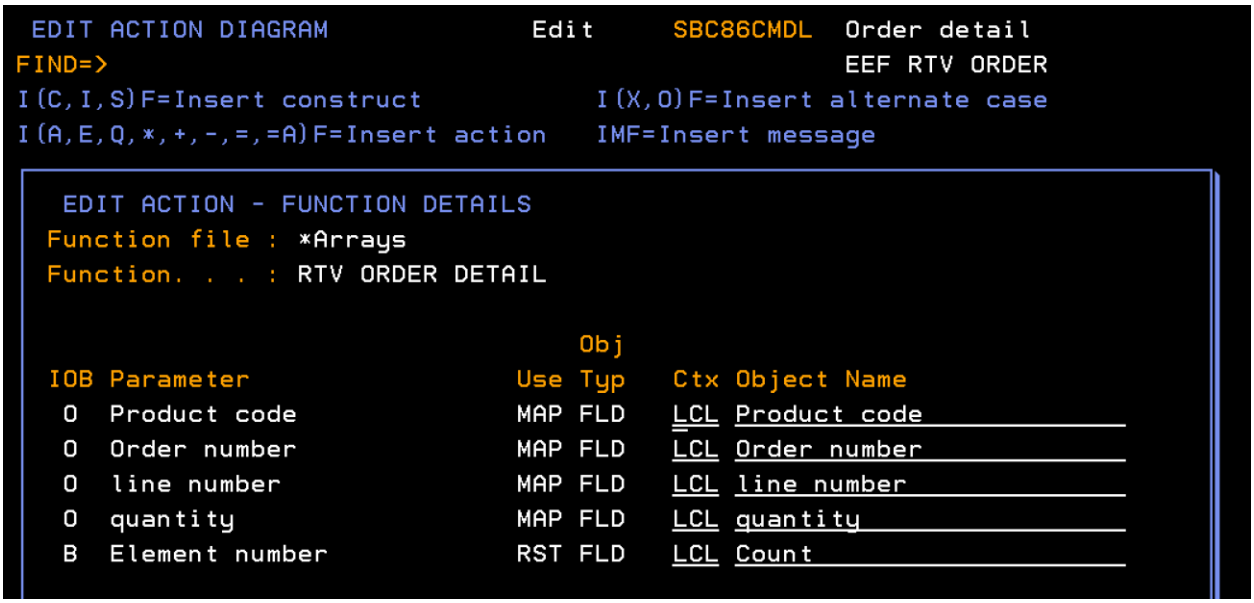
4. Initiate a call to the RTV ORDER HEADER as follows:



5. Initiate a call to LOAD ORDER DETAIL ARRAY as follows:



6. Initiate a call to RTV ORDER DETAIL as follows:



7. Save and exit the function.

Back to Top

Set the EXCEXTFUN to a Module

CA 2E automatically creates the EXCEXTFUN function type as PGM (Program), so you must change the type to MOD (Module).

Follow these steps:

1. From EDIT DATABASE RELATIONS take option F against the Order detail file to access the EDIT FUNCTIONS screen.
2. Take option Z (Details) against EEF RTV ORDER for the EDIT FUNCTION DETAILS screen.
3. Change the Target HLL to RP4 or CBI. Take option T (T-ILE Compilation Type) to change the object type to MOD.
4. Press enter to save the changes and press F3 to return the EDIT FUNCTIONS screen.

The EXCEXTFUN has all the necessary configuration and details, so you must then generate and compile before you can use it.

Back to Top

Generate and Compile the Module

With the EXCEXTFUN fully configured, you must compile the module.

Note: Before generating the module, your model must be set up to include PCML in the module. For more information, see the section “PCML in Module” in the Building Applications guide for details

Follow these steps:

1. Take option J (Generate) on the new EEF RTV ORDER function.
2. Press F17 for the Services Menu, and take Option 1 (Submit Model Create Request).
3. Locate the EEF RTV ORDER function and press Enter.

   Note: Depending on your processing speed, the compile can complete quickly or take several minutes.
4. Verify that the compile completed correctly.
5. Press F3 until you return the EDIT FUNCTIONS screen.

This procedure completes your work on the EXCEXTFUN module. Next, you create a Service Program for use with the module.

Back to Top

Create a Service Program

In the CA 2E model, you define a Service Program (*SRVPGM) function type, which you will then use with the module. You can customize the source member name, which is then used for the final *SRVPGM object name.

Follow these steps:

1. From EDIT DATABASE RELATIONS take option F against the Order detail file to access the EDIT FUNCTIONS screen.
2. Create a Service Program and complete the fields as follows:

   Function:

   Order srvpgm

   Function Type:

   Service Program

   Access path

   *NONE

A Service Program does not contain parameters or an action diagram, so you must add the module in the following procedure.

Back to Top

Add the Module to the Service Program

You now have your EEF RTV ORDER module and your Order srvpgm Service Program. Combine the two by adding the module to the Service Program.

Follow these steps:

1. Take option F against the Order detail file from EDIT DATABASE RELATIONS to access the EDIT FUNCTIONS screen.
2. Take option Z (Details) on the new service program Order srvpgm.
3. Take option M (Modules).
4. Press F6 (add) to add a module.
5. Take option X (Select) against the EEF RTV ORDER module.
6. Press F3 to access the SERVICE PROGRAM MODULE screen.

   Verify that the module is listed.
7. Press F3 until you return the EDIT FUNCTIONS screen.

The combined Service Program and module are not usable until you generate and compile them in the following procedure.

Back to Top

Generate and Compile the Service Program

The combined module and Service Program are not usable until you generate and compile.

Follow these steps:

1. Take option F against the Order detail file from EDIT DATABASE RELATIONS to access the EDIT FUNCTIONS screen.
2. Take option J (Generate) on the Order srvpgm Service Program.
3. Press F17 for the Services Menu, and take option 1 (Submit model create).
4. Press Enter to submit the job, which generates and compiles the source.

   Note: Depending on your processing speed, the compile can complete quickly or take several minutes.
5. Press F3 until you return to the EDIT FUNCTIONS screen.

This gives you a usable, compiled Service Program that includes the EEF RTV Order module you created. Next, create a Web Service Function that includes the compiled Service Program.

Back to Top

Create a Web Service Function

The CA 2E Web Service uses the Service Program you created in the previous procedures to create a Web Service Function. This function contains one-to-multiple operations, where each operation corresponds to a single procedure in a module in an ILE Service Program.

Follow these steps:

1. Access the EDIT FUNCTIONS screen.
2. Add a new Web Service function type by competing the following fields:

   Function

   Order websrv

   Function type

   Web Service

   Access Path

   *NONE
3. Press Enter to save, and then take option Z (Details) for the EDIT FUNCTION DETAILS screen.
4. Complete the details fields as follows:

   Web service name

   GetOrder

   Service program file

   Order Detail

   Service Program Function

   Order srvpgm
5. Press Enter to save, and then F3 until you return the EDIT FUNCTIONS screen.

The result is a modeled Web Service Function, but it is not available to invoke until you complete the following procedure to deploy Web Service Function on your Web Server.

Back to Top

Deploy the Web Service Instance

From the newly created Web Service Function, you can deploy a Web Service instance. Use the YCRTWS (Create Web Service Instance) command to install the web service to the IBM Web Services Server.

Notes:

- Before you deploy the Web Service Instance, verify that the Runtime Library List contains all of the libraries that are required to support the runtime business logic.
- Here we use the EDIT FUNCTION DETAILS screen, but you can also invoke YCRTWS from the command line and then configure the appropriate parameters.

Follow these steps:

1. Take option F against the Order detail file from EDIT DATABASE RELATIONS to access the EDIT FUNCTIONS screen.
2. Take option Z against the web service Order websrv.
3. Press F10 (Create Web Service Instance) and complete the fields as follows:

   Update Model

   *ADD

   Install to Server

   *YES

   Web Services Server

   YourServerName (the actual name for you Web Server)
4. Press Enter to save and deploy.

The Web Service Function is now available for users to call from your Web Server. Test the Web Service before notifying your users.

For detailed information on web service deployment see the Chapter Web Service Creation (see page 191).

Note: Use the IBM Web Administration interface to start, stop, and test the deployed Web Service.

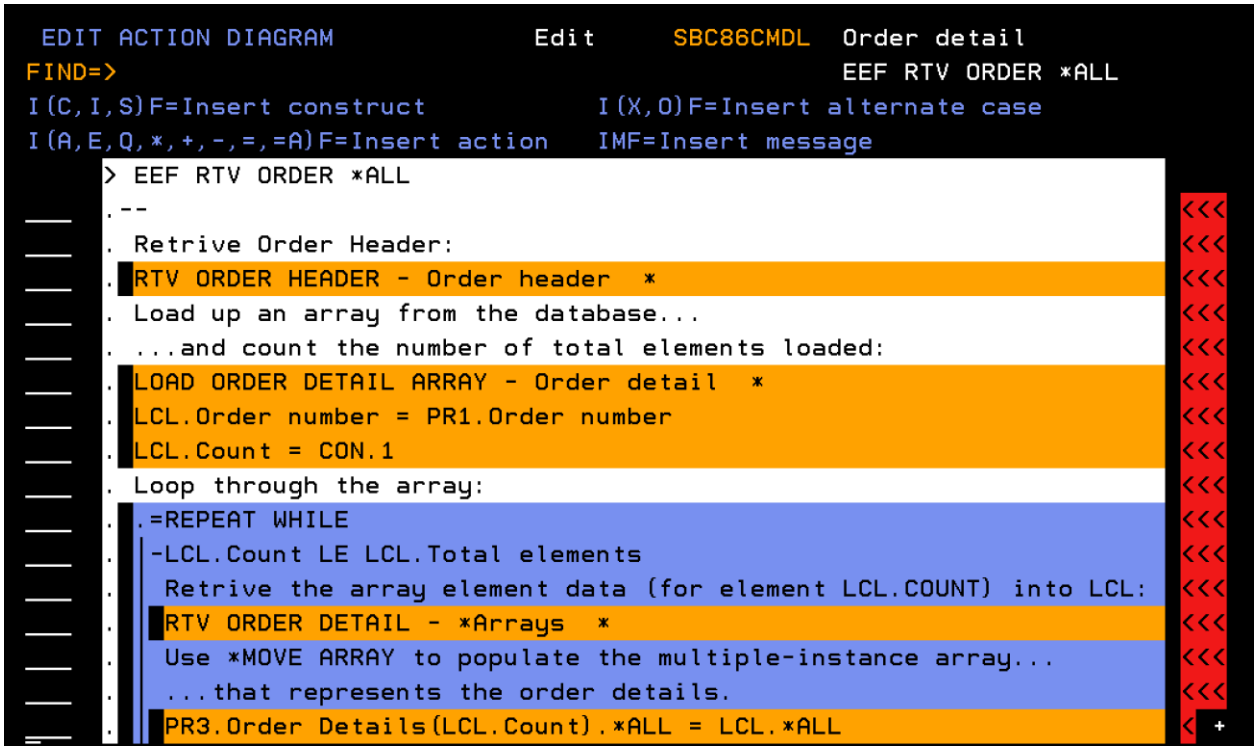
Back to Top

*MOVE ARRAY (*ALL)

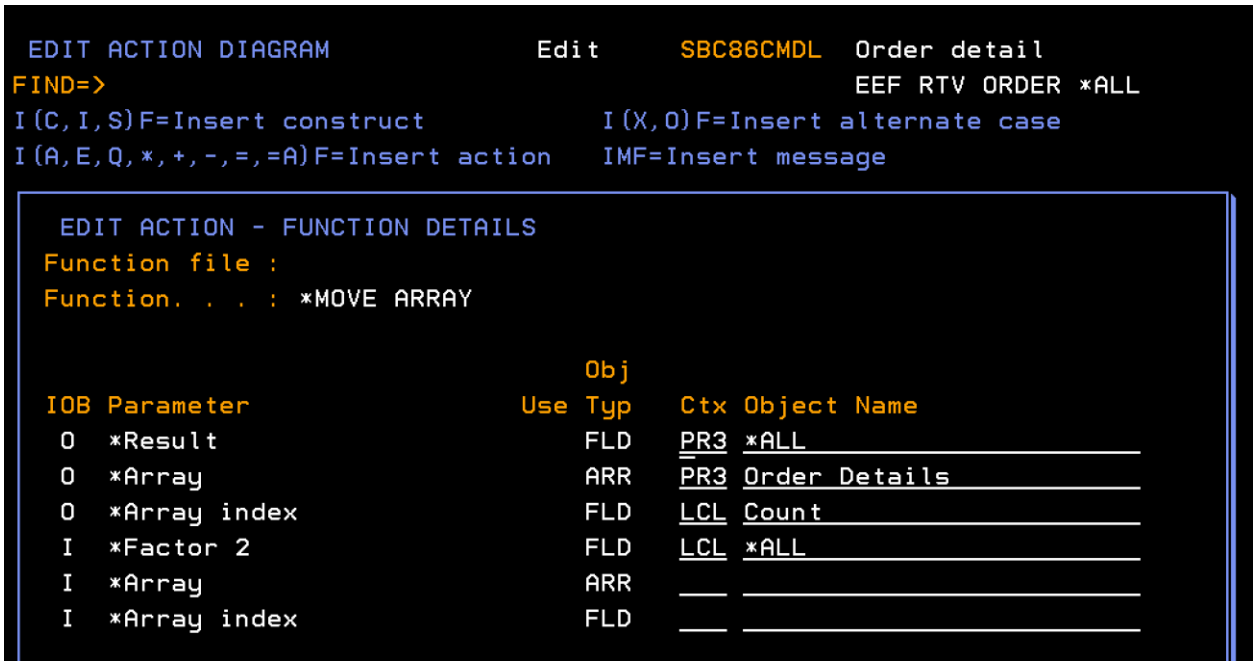
Four individual *MOVE ARRAY statements are used in the example code for EEF RTV Order (Order detail) (see page 757) to move data from a field in the LCL context to a field in a multiple instance array parameter. This is a good illustration of how you can use *MOVE ARRAY. However, you can use *MOVE ARRAY, *ALL support for a more efficient approach.

You can replace the multiple *MOVE ARRAY statements with a single *MOVE ARRAY statement, specifying *ALL for the *Result and *Factor fields. You do not have to change the action diagram if you then add or remove fields from the array; a simple regeneration and recompilation works.

The following EDIT ACTION DIAGRAM panel shows how you can use *MOVE ARRAY (*ALL:)



The following EDIT ACTION DIAGRAM – FUNCTION DETAILS panel shows more detail on how you can use *MOVE ARRAY (*ALL:):



Back to Top

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z